Spezifische Fehlertoleranz für kombinatorische und sequentielle Schaltungen

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

genehmigte Dissertation

vorgelegt von

Diplom-Informatiker

Michael Augustin

geboren am 9. Juli 1980 in Potsdam

Gutachter: Prof. Dr. Rolf Kraemer

Gutachter: Prof. Dr. Michael Gössel

Gutachter: Prof. Dr. Heinrich Theodor Vierhaus

Tag der mündlichen Prüfung: 12. Juli 2012

Kurzfassung

In dieser Arbeit wird ein neues Konzept für den Entwurf fehlertoleranter digitaler Schaltungen vorgestellt. Die als spezifische Fehlertoleranz bezeichnete Entwurfsmethode erweitert den Stand der Technik um die Möglichkeit, Fehlertoleranz gezielt für gewünschte Ein- und Ausgabezuweisungen einer Schaltung bereitzustellen. Die generell sehr aufwändigen Maßnahmen zur Realisierung von Fehlertoleranz lassen sich so an die tatsächlich vorhandenen Zuverlässigkeitsanforderungen einer gegebenen Anwendung anpassen, was im Vergleich zu herkömmlichen Verfahren zu erheblichen Kosteneinsparungen im Hinblick auf Hardware und Stromverbrauch führt.
Die Eingaben einer Schaltung werden dazu mit ihren zugehörigen Ausgaben in kritische und unkritische Signale unterteilt. Für kritische Signale wird das gleiche Maß an Fehlertoleranz bereitgestellt, wie es auch durch das Verfahren der dreifach modularen Redundanz garantiert wird. Unkritische Signale werden nicht fehlertolerant ausgelegt, da das für solche Signale nicht gefordert werden muss.
Im Bezug auf kombinatorische Schaltungen wird dieses Grundprinzip auf eine beliebig wählbare Teilmenge aller möglichen Eingaben mit ihren zugehörigen Ausgaben angewendet. Die nicht zu schützenden, unkritischen Signale werden zu Optimierungszwecken genutzt. Genau diese Freiheit im Entwurfsprozess ermöglicht beim Entwurf fehlertoleranter Schaltungen signifikante Kosteneinsparungen, die bislang durch andere Verfahren nicht berücksichtigt wurden.
Bei sequentiellen Schaltungen wird das Konzept auf Eingabefolgen und ihre entsprechend zugehörigen Ausgabefolgen abgebildet. Ab einem bestimmten Zustand werden in Schaltungen, die nach diesem Prinzip entworfen wurden, alle als kritisch eingestuften Eingabefolgen fehlertolerant verarbeitet, woraufhin die sequentielle Schaltung entsprechend fehlertolerante Ausgabefolgen liefert. Die Länge der kritischen Eingabefolgen und die Anzahl der Zustände, von denen aus eine fehlertolerante Verarbeitung der Eingaben gefordert wird, ist beliebig wählbar.
Neben der Beschreibung des Grundkonzeptes der spezifischen Fehlertoleranz für kombinatorische und sequentielle Schaltungen beinhaltet die Arbeit Erweiterungen, mit denen sich die durch das Verfahren bereitgestellte Fehlertoleranz erhöhen lässt. Es wird zudem erläutert, wie die spezifische Fehlertoleranz ohne besonderen Aufwand mit Hilfe gängiger Werkzeuge im Schaltungsentwurf umgesetzt werden kann. Vom Entwerfer werden dazu keine speziellen Kenntnisse über das eigentliche Verfahren vorausgesetzt. Anhand experimenteller Ergebnisse wird auch gezeigt, welche Einsparungen sich durch die spezifische Fehlertoleranz im Vergleich zu herkömmlichen Verfahren aus diesem Bereich ergeben. Dazu wurden verschiedene Benchmark-Schaltungen mit der spezifischen Fehlertoleranz implementiert und ihre Gesamtflächen den Flächen entsprechender Schaltungen, die nach dem Prinzip der dreifach modularen Redundanz entworfen wurden, gegenübergestellt.

Abstract

This thesis introduces a new concept for the design of fault-tolerant digital circuits. It is called selective fault tolerance and it improves the state of the art by providing the possibility to specifically implement fault tolerance on a selected set of signals of a given circuit. The benefit is a significant reduction in the additional cost that is usually required by conventional design methods from this field of research. Since selective fault tolerance allows to adapt the feature of fault tolerance to the real requirements of a given application, the hardware overhead and the additional power consumption of redundant components can be significantly reduced.

To realize this concept, the set of input and output assignments is divided into critical and uncritical signals. Critical signals will be processed with the same degree of fault tolerance as it is done in circuits equipped with triple modular redundancy. Uncritical signals will not be implemented with fault tolerance, because the application does not require it.

In case of combinational circuits, the basic concept of selective fault tolerance is applied to an arbitrarily selectable subset of all possible input assignments and their associated output assignments. The uncritical signals which do not need to be implemented with fault tolerance will be utilized for optimization purposes. This freedom in the design process allows for significant cost reduction which has not been considered previously by other methods.

With respect to sequential circuits, the basic concept of selective fault tolerance is mapped to input sequences and their corresponding output sequences. Starting from a certain state, every input sequence that is classified to be critical will be processed fault-tolerantly. Thus, the corresponding output sequence is generated fault-tolerant as well. The number of states that are initial states for a critical input sequence and also the length of the sequences to be protected can be arbitrarily chosen by the designer.

In addition to the basic concept of selective fault tolerance for combinational and sequential circuits, further ideas are presented which improve the fault tolerance that is provided by the proposed method. Furthermore, a technique of implementing selective fault tolerance with conventional design tools without any notable effort is introduced. To this end, the designer does not even require special knowledge about the actual algorithm of selective fault tolerance. It is sufficient that the designer combines some netlists.

The experimental results that will be presented in this thesis are based on a selection of several benchmark circuits. These benchmark circuits were implemented according to the concept of selective fault tolerance for a variable number of input and output assignments and also according to the concept of triple modular redundancy. The results obtained were compared with regard to the additional hardware overhead of each method.

Vorwort

Der Entwurf fehlertoleranter elektronischer Systeme ist nicht mehr nur ein Spezialgebiet für Anwendungsbereiche, in denen Schaltungen hoher radioaktiver Strahlung oder anderen intensiven Störgrößen aus ihrer Umgebung ausgesetzt sind. Die Anzahl der Fehler, die auch in technischen Geräten beobachtet werden, obwohl sie nicht unter solchen extremen Bedingungen betrieben werden, nimmt permanent zu. Ein kurzer Überblick über mögliche Fehlerursachen wird in der Einleitung dieser Arbeit gegeben.
Die Fehler und ihre Auswirkungen müssen unter allen Umständen vermieden werden. Das gilt insbesondere dann, wenn elektronische Schaltungen kritische Aufgaben steuern, von denen z.B. Menschenleben abhängen. Zahlreiche solcher Anwendungen finden sich im Transportwesen, der Medizintechnik oder der Automatisierungsindustrie. Da elektronische Schaltungen aber auch zunehmend in anderen Bereichen zur Realisierung wichtiger Aufgaben eingesetzt werden, gewinnt der Aspekt der Zuverlässigkeit für die Elektronik immer mehr an Bedeutung.
Die Kosten für den Entwurf zuverlässiger elektronischer Systeme sind hoch. Das betrifft sowohl die Umsetzung auf Hardware- als auch auf Softwareebene. Egal ob Fehler vermieden, toleriert oder behoben werden sollen, es müssen immer zusätzliche Komponenten eingesetzt oder Berechnungen durchgeführt werden, um diese Prinzipien umzusetzen. Eine wesentliche Zielstellung für den Entwurf solcher Systeme besteht daher darin, den Aufwand für Maßnahmen zur Fehlerkompensation so gering und effektiv wie möglich zu gestalten.
In dieser Arbeit wird eine Möglichkeit vorgestellt, mit der die Kosten für den Entwurf fehlertoleranter Digitalschaltungen gesenkt werden können. Im Gegensatz zu herkömmlichen Verfahren wird dazu die Fehlertoleranz gezielt auf die real vorhandenen Anforderungen einer gegebenen Anwendung angepasst, wodurch sich ein erhebliches Maß an Hardwareredundanz einsparen lässt.
Das in diesem Zusammenhang vorgestellte Verfahren wurde in verschiedenen Vorarbeiten zu dieser Dissertation [AGK10c, AGK10b, AGK10a, AGK11a, AGK11b, AGK11c, AGSK12] veröffentlicht. Sie entstanden durch die außerordentlich gute Zusammenarbeit mit Prof. Dr. Rolf Kraemer von der Brandenburgischen Technischen Universität Cottbus und Prof. Dr. Michael Gössel von der Universität Potsdam, denen ich an dieser Stelle für ihre umfangreiche Hilfe danken möchte.
Gerade erst die wegweisende wissenschaftliche Unterstützung, die ich von Prof. Dr. Michael Gössel erfahren durfte und die große Freiheit, die mir Prof. Dr. Rolf Kraemer zur Realisierung meiner Forschungsarbeiten an seinem Lehrstuhl sowie in seiner Abteilung am Leibniz-Institut für innovative Mikroelektronik in jeder Hinsicht gewährt hat, haben es mir überhaupt ermöglicht, die vorliegende Dissertation anzufertigen.
Zudem bedanke ich mich bei Prof. Dr. Heinrich Theodor Vierhaus und seinen Mit-

arbeitern für den kontinuierlichen Wissensaustausch. Sie haben mir das benachbarte Forschungsgebiet der eingebauten Selbstreparatur nahegebracht und mein Verständnis für die Fehlerkorrektur in digitalen Schaltungen erheblich erweitert.
Weiter möchte ich auch meine Kollegen vom Leibniz-Institut für innovative Mikroelektronik erwähnen, die mir ein wunderbares Forschungsumfeld geschaffen haben und mir jederzeit mit Rat und Tat zur Seite standen. Mein besonderer Dank gilt in diesem Zusammenhang Steffen Zeidler, Marcus Ehrig, Vladimir Petrovic, Gunter Schoof und Dr. Miloš Krstić.
Ebenso danke ich auch meinen Eltern für das kurzfristige Korrekturlesen von weiten Teilen dieser Arbeit.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Die Halbleiterindustrie steht ständig vor der Herausforderung schnellere, komplexere und stromsparendere Chips für verschiedenste Anwendungsbereiche zu entwickeln. Um diesen Anforderungen immer wieder gerecht werden zu können, verbessern die Hersteller permanent die Technologien für ihre Produkte. Neben der Erforschung neuer Materialien steht dabei besonders die Skalierung der Strukturgrößen elektronischer Bauteile im Vordergrund. Aktuell liegen die Größenordnungen bei den Marktführern im Bereich von etwa 20 Nanometern. Diese kleinen Strukturen haben nicht nur den Vorteil, dass sie sehr wenig Platz einnehmen. Sie ermöglichen auch höhere Schaltgeschwindigkeiten und weisen zudem einen geringeren Stromverbrauch auf [Bor99].
Die Einführung einer neuen Technologie bringt jedoch auch Probleme mit sich, die einerseits den Fertigungsprozess und andererseits die Hardware selbst betreffen. Sie werden in der Literatur durch zahlreiche Defektmechanismen und Fehlermodelle beschrieben, die die Grundlage für verschiedene Lösungsmöglichkeiten auf unterschiedlichen Abstraktionsebenen sind. Auszüge aus der zu diesen Themengebieten umfangreich existierenden Literatur werden nachfolgend in einem kurzen Überblick zusammengefasst. Dieser Überblick ist eine wichtige Basis für das in dieser Dissertation vorgestellte Verfahren zur Fehlerkorrektur in digitalen Schaltungen. Er zeigt die Notwendigkeit für den Entwurf zuverlässiger elektronischer Systeme und beinhaltet eine Übersicht verschiedener Lösungsansätze, mit denen dieses Ziel umgesetzt werden kann.
Eine besondere Schwierigkeit bei der Herstellung moderner Halbleiterprodukte ist die fehlerfreie Erzeugung der extrem kleinen und nahe beieinander liegenden Strukturen. Sie werden durch chemische, optische und mechanische Prozesse auf einen Wafer aufgetragen und müssen äußerst exakt dimensioniert sein. Diese notwendige Präzision wird in der Realität jedoch nicht erreicht, obwohl der Fertigungsprozess mit jeder Verkleinerung der Strukturen immer genauer werden muss. So bleiben nach der Herstellung z.B. Rückstände aus der Fertigung auf der Schaltung, Leiterbahnen liegen zu nahe beieinander, sind stellenweise zu schmal oder zu breit oder Verbindungsstrukturen passen nicht genau aufeinander. Aufgrund dieser Problematiken werden in den Schaltungen zunehmend Defektmechanismen und Instabilitäten erzeugt, die bei größeren Technologien noch nicht von Bedeutung waren oder aufgetreten sind [Ait04]. Einige davon führen direkt nach der Herstellung zu Fehlern, andere wirken sich erst nach einiger Zeit oder unter bestimmten Bedingungen wie extremen Temperaturen aus [KOWS06].
Produktionsbedingte Fehler können von den Herstellern durch Anpassung des Fertigungsprozesses behoben werden. Eine Schaltung ist zur Laufzeit aber auch dem Einfluss verschiedener Störgrößen aus ihrer Umgebung ausgesetzt. Diese können ebenfalls Auslöser eines Fehlers sein, da schon geringste Abweichungen in den Betriebsparametern dazu führen können, dass Schaltungen im laufenden Betrieb nicht mehr ihrer ur-

sprünglichen Spezifikation entsprechen. Häufig treten in diesem Zusammenhang Störungen durch Alpha-Partikel, die sich aus Verunreinigungen im Verpackungsmaterial lösen, oder radioaktive Strahlung auf [Bau05]. Ebenso sind elektromagnetische Interferenzen, die z.B. durch das gleichzeitige Schalten angrenzender Schaltungsteile erzeugt werden, Ursachen für Fehler [AGMR02]. Diese Problematiken verschärfen sich mit der Erhöhung der Integrationsdichte und der Einführung kleinerer Technologien [CA08]. Da die aktuellen Schaltungen mit nur noch sehr niedrigen Spannungspegeln betrieben werden, reicht schon eine geringe energetische Störung aus, um die in einer Speicherzelle vorgehaltene Ladung zu ändern oder Schwankungen der Betriebsparameter von Schaltungen hervorzurufen, die dann ebenfalls zu Fehlern führen. Durch die kleinen Abstände in den Schaltungen wird zudem das Übersprechen benachbarter Leitungen begünstigt.
Hinzu kommen Stresseffekte wie die Hot Carrier Injection oder Negative Bias Temperature Instability, die mit zunehmender Verkleinerung der Technologien und unproportionaler Skalierung der Versorgungsspannung zu einer schnelleren Abnutzung und somit frühzeitig zu Ausfällen in der zunächst defektfreien Schaltungen führen [Sch10]. Zu diesen Degradationsprozessen gehört auch die Elektromigration, bei der durch frei bewegliche Elektronen in den ungleichmäßig dimensionierten Leiterbahnen Metallatome von einem Ort zum anderen transportiert werden [AVU+08]. So entstehen, je nachdem ob zu viel oder zu wenig Material an einer bestimmten Stelle einer Leitung ist, offene Kontakte oder Kurzschüsse, die dazu führen können, dass eine Schaltung nicht mehr ihrer funktionalen Spezifikation entspricht.
Die Fehler werden nach [Con02] entsprechend ihrer Art und Auswirkung in die Kategorien permanent, intermittierend und transient unterteilt. Permanente Fehler treten in einer Schaltung ständig an derselben Stelle auf und werden durch einen dauerhaften Defekt der Schaltung hervorgerufen. Der Defekt kann durch Prozessungenauigkeiten oder Verunreinigungen bei der Herstellung, Alterungseffekte oder die genannten Einflüsse aus der Umgebung verursacht werden.
Intermittierende Fehler haben ähnliche Ursachen wie permanente Fehler und treten ebenfalls immer wieder an derselben Stelle einer Schaltung auf. Ihr Auftrittszeitpunkt ist allerdings variabel, da sie durch Instabilitäten der Schaltung ausgelöst werden. Eine Instabilität kann eine Vorstufe eines Defektes sein oder dadurch verursacht werden, dass eine Schaltung nahe an oder über den Grenzen ihrer Leistungsfähigkeit betrieben wird. Entsprechend zeigen sich intermittierende Fehler nur unter bestimmten Bedingungen oder wenn ihr Auftreten durch zusätzliche Einflussfaktoren begünstigt wird.
Transiente Fehler treten nur vorübergehend in einem von einer Störgröße betroffenen Schaltungsteil auf. Sie werden entsprechend ihrer Auswirkungen auf verschiedene Bauteile in weitere Gruppen unterteilt. Gemäß [Nic10] wird in diesem Zusammenhang eine durch eine Störgröße verursachte Änderung des logischen Wertes in einer Speicherzelle, die bis zum nächsten Ladevorgang oder nächsten Reset erhalten bleibt, als Single Event Upset (SEU) bezeichnet. Folgefehler, die z.B. durch einen kurzzeitigen Fehler im Zustandsspeicher eines Automaten ausgelöst werden und mit dem nächsten Reset behoben werden können, werden Single Event Functional Interrupts (SEFI) genannt. Ein Single Event Latch-up (SEL) ist ein Fehler in einem Transistor, bei dem dieser durch das Auslösen parasitärer PNPN Strukturen ungewollt in einen niederohmigen Zustand mit hohem

Stromfluss versetzt wird. Spannungsschwankungen in kombinatorischen Schaltungsteilen werden als Single Event Transient (SET) bezeichnet.
Im Vergleich zur kombinatorischen Logik sind die Folgen transienter Fehler in Speicherelementen gravierender. Da ein SET nach kurzer Zeit automatisch ausgeglichen wird, ist er nur dann problematisch, wenn er sich genau zu dem Zeitpunkt auf die Ausgabe der kombinatorischen Logik auswirkt, zu dem diese Ausgabe weiterverarbeitet oder durch ein Speicherelement abgespeichert werden soll. Hinzu kommt, dass sich transiente Fehler in kombinatorischer Logik nicht grundsätzlich durch die gesamte Schaltung bis hin zum Ausgang ausbreiten. Einerseits können sie logisch und andererseits elektrisch von anderen kombinatorischen Schaltungsteilen maskiert werden [SKK+02]. Simulationen zeigen aber, dass diese Eigenschaften mit kleineren Technologien und höheren Taktraten nachlassen werden [BBB+97]. Selbst wenn die Schwankungen nur sehr kurz auftreten, tragen die geringen Verzögerungen der kleinen Technologien und die hohe Taktraten dazu bei, dass sich die Fehler weit durch die Schaltung bis hin zum Ausgang oder ein nachfolgend geschaltetes Speicherelement ausbreiten können. Durch die schnell hintereinander folgenden Speichervorgänge haben die Fehler in schnelleren Schaltungen zusätzlich eine höhere Wahrscheinlichkeit genau zum Zeitpunkt der Abspeicherung am Ausgang der kombinatorischen Logik vorzuliegen [DNR02]. Transiente Fehler in Speichern bleiben hingegen so lange erhalten, bis der aktuelle Wert des Speichers überschrieben wird. In dieser Zeit kann die Weiterverarbeitung des Fehlers fatale Folgen für die Anwendung der Gesamtschaltung haben.
Die genannten Beispiele sind lediglich ein Auszug aus den zahlreichen Problemen, die mit der Einführung neuer Technologien berücksichtigt werden müssen. Sie zeigen, dass der Einsatz von Methoden zur Erhöhung oder Aufrechterhaltung der Zuverlässigkeit elektronischer Schaltungen ein wesentliches Gebiet zukünftiger Entwicklungen in der Halbleitertechnik sein wird. Der Entwurf zuverlässiger Systeme ist somit nicht mehr nur für kritische Anwendungen oder Bereiche, in denen die Elektronik sehr extremen Bedingungen ausgesetzt ist, von Interesse, sondern gewinnt auch zunehmend an Bedeutung für die Herstellung von Massenprodukten.
Um die technologiebedingt zunehmend unzuverlässigeren Schaltungen weniger anfällig für Fehler zu machen, werden verschiedene Strategien auf den Ebenen Prozesstechnologie, Schaltungsentwurf und Schaltungsarchitektur verfolgt [Mas96]. Hierbei muss zwischen Maßnahmen zur Fehlervermeidung, Fehlertoleranz und Fehlerbeseitigung unterschieden werden.
Zur Fehlervermeidung zählen Abschirmungsmaßnahmen und technologische Verbesserungen, die die Schaltungen widerstandsfähiger gegenüber Umwelteinflüssen machen. Aber auch die Verstärkung von Strukturen, Anpassungen im Layout zur Reduzierung des Einflusses anderer Schaltungsteile oder die Verwendung robusterer Speicherelemente mit einer größeren Anzahl an Transistoren fallen in diesen Bereich. Sie können allerdings nur die Auftrittswahrscheinlichkeit eines Fehlers reduzieren. Einen Fehler korrigieren können diese Techniken nicht.
Dieses Ziel verfolgen jedoch Methoden aus dem Bereich der Fehlertoleranz. Sie beinhalten die nebenläufige oder zeitlich verzögerte Korrektur eines Fehlers während des laufenden Betriebs einer Schaltung. Die Korrektur eines Fehlers kann durch zusätzli-

che Hardware, zusätzliche Berechnungen oder einer Mischung aus beidem durchgeführt werden. In jedem Fall wird aber irgendeine Form von Redundanz benötigt, um den Fehler entweder zu maskieren oder ihn zu erkennen und anschließend zu korrigieren. Die Fehlerkorrektur durch redundante Hardware ist wesentlicher Bestandteil dieser Arbeit. Gängige Methoden, die in diesen Bereich fallen und auch praktisch eingesetzt werden, sind z.B. die Systemverdreifachung mit Mehrheitsentscheid, die Fehlererkennung mit Umschaltung und fehlerkorrigierende Kodes [Joh88, Pra96, Lal01, HP03].
Die Fehlerbeseitigung wird durch Verfahren der eingebauten Selbstreparatur vorgenommen. Im Gegensatz zu Fehlertoleranzmethoden verfolgen Techniken aus diesem Bereich das Ziel, einen von einem Fehler betroffenen Schaltungsteil über einen eingebauten Selbsttest zu identifizieren, um ihn anschließend vom weiteren Betrieb auszuschließen. Sie eignen sich daher insbesondere zur Beseitigung permanenter Fehler, die auf einen Defekt zurückzuführen sind und Maßnahmen zur Fehlertoleranz dauerhaft ausnutzen würden. Eine fehlertolerante Schaltung kann also auch durch Verfahren der Selbstreparatur ergänzt werden, um im Fall eines permanenten Fehlers die ursprüngliche Fehlertoleranz wieder herzustellen. Praktisch wird die eingebaute Selbstreparatur vor allem in Speichern eingesetzt, um nachträglich Fertigungsfehler zu korrigieren [KZK+98]. Für diese regelmäßig aufgebauten Strukturen können redundante Systemteile sehr effektiv bereitgestellt und im beschriebenen Fehlerfall aktiviert werden, sobald die defekten Teilschaltungen deaktiviert wurden. Bei irregulären Strukturen, wie kombinatorischer Logik, gestaltet sich diese Vorgehensweise problematischer. Einerseits ist es Ziel der Selbstreparatur, den Austausch von defekten mit redundanten Schaltungsteilen auf möglichst fein-granularer Ebene durchzuführen, andererseits können aber nicht beliebig viele zusätzliche irreguläre Teilschaltungen bereitgestellt werden, um grundsätzlich eine ideale Ersetzungsstrategie zu ermöglichen. Eine Lösung für diese Problematik wird in [GKV10] durch einen Clustering-Algorithmus beschrieben, der aus den irregulären Logikstrukturen reguläre Teilmengen bestimmt. Diese können dann nach dem bewährten Prinzip ersetzt werden. Darüber hinaus ist die eingebaute Selbstreparatur auch in Verbindung mit rekonfigurierbaren Systemen wie Field Programmable Gate Arrays (FPAGs) vorteilhaft einsetzbar.
Die genannten Verfahren sind immer mit zusätzlichen Kosten verbunden, welche dadurch gesenkt werden können, dass die Behandlung von Fehlern nur auf bestimmte Teile einer Schaltung beschränkt wird. Dieses Vorgehen wird unter dem Begriff Selective Hardening zusammengefasst [PH11]. Dabei werden strukturelle Modifikationen in den Schaltungen vorgenommen, die allerdings nicht die eigentliche Funktionalität der Schaltungen berücksichtigen.
Eine neue Fehlertoleranzmethode, die funktional genau auf die Anforderungen einer gegebenen Anwendung angepasst werden kann, wird in dieser Arbeit vorgestellt. Die Methode nutzt aus, dass in den meisten praktischen Anwendungen nicht alle Signale geschützt werden müssen, da nur eine Teilmenge aller Signale tatsächlich zur Realisierung kritischer oder systemrelevanter Aufgaben notwendig ist. Als kritisch werden in diesem Zusammenhang Aufgaben bezeichnet, von denen z.B. Menschenleben oder Missionen abhängen. Unkritisch sind alle anderen Aufgaben, die im Fehlerfall nicht die grundlegende Funktionalität der eigentlichen Anwendung beeinträchtigen. Ein Beispiel wäre das

Airbag System eines Autos im Vergleich zur Zentralverriegelung. Welche Signale konkret geschützt werden sollen, kann vom Entwerfer selbst bestimmt werden. Genau dadurch ergibt sich auch der Vorteil der Methode. Da die Fehlertoleranz nur für bestimmte Signale bereitgestellt wird, kann gegenüber herkömmlichen Verfahren ein erheblicher Anteil an benötigter Redundanz eingespart werden. Der Grad an Fehlertoleranz ist für die geschützten Signale identisch zur Systemverdreifachung.
Die Methode wird als spezifische Fehlertoleranz bezeichnet. Sie wird in dieser Dissertation sowohl für kombinatorische als auch für sequentielle Schaltungen ausgearbeitet. Im Zusammenhang mit kombinatorischen Schaltungen ermöglicht sie die gezielte Umsetzung von Fehlertoleranz auf einer sogenannten kritischen Teilmenge aller möglichen Ein- und Ausgabezuweisungen. Um Kosten bezüglich der zusätzlich benötigten Hardware einzusparen, werden alle übrigen, unkritischen Signale nicht fehlertolerant ausgelegt. Dieses Prinzip wird in sequentiellen Schaltungen auf Ein- und Ausgabefolgen abgebildet. Es wird dazu angenommen, dass eine sequentielle Schaltung nur dann fehlertolerant verarbeitete Ausgaben liefern muss, wenn sie sich in bestimmten ausgezeichneten Zuständen befindet. Der Übergang von einem beliebigen Zustand in einen ausgezeichneten Zustand wird zunächst nicht geschützt. Es wird aber im Verlauf der Arbeit gezeigt, dass die Verwendung fehlertoleranter Zustandsspeicherelemente genau diese Funktionalität übernehmen kann. Die Ausgabefunktion bleibt dabei weiterhin nur für eine Teilmenge aller Ein- und Ausgabenfolgen fehlertolerant. Die allgemeinen Definitionen der spezifischen Fehlertoleranz werden für kombinatorische und sequentielle Schaltungen anhand einfacher Beispiele erklärt. Ihre Anwendbarkeit im realen Schaltungsentwurf wird anhand von Benchmark-Schaltungen präsentiert und Einsparungen, die sich in diesem Zusammenhang bezüglich der zusätzlichen Hardwarekosten im Vergleich zur Systemverdreifachung ergeben, werden ebenfalls dargestellt und kommentiert.
Da es ein besonderes Ziel dieser Dissertation ist, die vorgestellte Methode der spezifischen Fehlertoleranz auch für praktisch relevante Systeme anwendbar zu machen, werden verschiedene Algorithmen vorgeschlagen, durch die die Methode problemlos mit Hilfe gängiger Werkzeuge im Schaltungsentwurf umgesetzt werden kann. Auf diese Weise muss sich ein Entwerfer nicht in die Einzelheiten des Verfahrens einarbeiten, sobald er eine Schaltung nach diesem Prinzip entwerfen möchte.
Nach diesem einleitenden ersten Kapitel ist die Dissertation ist in folgende weitere Abschnitte unterteilt. In Kapitel 2 werden die grundlegenden Fehlertoleranzmethoden nach dem bisher gegebenen Stand der Technik beschrieben. In Kapitel 3 wird die neue, darauf aufbauende Methode der spezifischen Fehlertoleranz für kombinatorische Schaltungen vorgestellt. In Kapitel 4 werden an Beispielen verschiedene Möglichkeiten gezeigt, nach denen die Methode praktisch für beliebige kombinatorische Schaltungen implementiert werden kann. Dabei stehen Kriterien wie die möglichst optimale Kostenreduzierung und praktische Umsetzbarkeit mit gängigen Entwurfswerkzeugen im Vordergrund. Ihre Effektivität im Bezug auf die Flächenreduzierung wird im Vergleich zur Systemverdreifachung mit Mehrheitsentscheid anhand experimenteller Ergebnisse präsentiert. In Kapitel 5 wird gezeigt, wie das grundlegende Konzept der spezifischen Fehlertoleranz für kombinatorische Schaltungen im Hinblick auf sequentielle Schaltungen verallgemeinert werden kann. Auch in diesem Zusammenhang werden verschiedene Entwurfsstrategien,

z.B. in Verbindung mit weiteren Fehlertoleranzmethoden, vorgestellt, deren Möglichkeit zur Kostenreduzierung mit Hilfe experimenteller Ergebnisse belegt wird. Kapitel 6 enthält eine Zusammenfassung der Arbeit.

2. Stand der Technik auf dem bearbeiteten Forschungsgebiet

In diesem Kapitel werden bekannte Entwurfstechniken zur Fehlerkompensation in digitalen Schaltungen vorgestellt. Sie gehören zum Stand der Technik und werden neben den angegebenen Quellen ausführlich auch in weiterer Fachliteratur beschrieben. Ausgangspunkt bildet für jedes Verfahren eine zu schützende Schaltung S mit m Eingängen und n Ausgängen, welche die Boolesche Funktion $S(x) = y$ realisiert. Dabei sind $m, n \in \mathbb{N}^+$ und S kann sowohl eine kombinatorische als auch sequentielle Schaltung sein.

2.1. Systemvervielfachung mit Mehrheitsentscheid

Die einfachste Variante der Systemvervielfachung mit Mehrheitsentscheid nach [Neu56] wird auch als Triple Modular Redundancy (TMR) bezeichnet [Joh88, Pra96]. Zur Umsetzung dieses Konzeptes wird die zu schützende Schaltung S verdreifacht und die Ausgänge der drei identischen Schaltungen werden in die Eingänge eines Mehrheitsentscheiders V geführt. Alle drei Schaltungen erhalten gleichzeitig dieselben m-stelligen Eingabevektoren $x \in \{0,1\}^m$ und erzeugen im fehlerfreien Fall an ihren Ausgängen gleichzeitig dieselben n-stelligen Ausgabevektoren $y_1 = y_2 = y_3 \in \{0,1\}^n$. Der Mehrheitsentscheider, der im Folgenden auch als Voter bezeichnet wird, bildet komponentenweise aus der Mehrheit seiner Eingaben den n-stelligen Ausgabevektor y. Die Funktion, die der Voter zur Bestimmung einer Komponente k des Ausgabevektors $y = y^1, \ldots, y^n$ ausführt, ist definiert durch $y^k = y_1^k y_2^k \vee y_1^k y_3^k \vee y_2^k y_3^k$. Der Systemaufbau einer solchen TMR-Schaltung ist in Abbildung 2.1 dargestellt.
Eine Schaltung, die nach diesem Prinzip entworfen wurde, toleriert alle Fehler, die sich komponentenweise maximal auf den Ausgabevektor einer der drei identischen Schaltungen auswirken. Neben dem sehr einfachen Systemaufbau und der fast verzögerungsfreien Fehlerkorrektur durch den Voter ist es zusätzlich auch von Vorteil, dass für diese Technik kein spezielles Fehlermodell angenommen werden muss, für das die Schaltung fehlertolerant ist. Von Nachteil ist der große Hardwareaufwand, denn die resultierende Schaltung nimmt mehr als 300% der ursprünglichen, nicht fehlertoleranten Schaltung ein und hat einen entsprechend höheren Stromverbrauch, da die zusätzlichen Schaltungsteile ebenfalls mit Strom versorgt werden müssen. Außerdem stellt der Voter den Schwachpunkt dieser Systemarchitektur dar. Ein Fehler in diesem Schaltungsteil wird von dem Systemaufbau nicht toleriert und kann sich am Ausgang des Voters auswirken. Durch die Verdreifachung der Voter selbst kann diese Problematik innerhalb der Schaltung umgangen werden. Spätestens am Ausgang der Gesamtschaltung hängt die Ausgabe jedoch

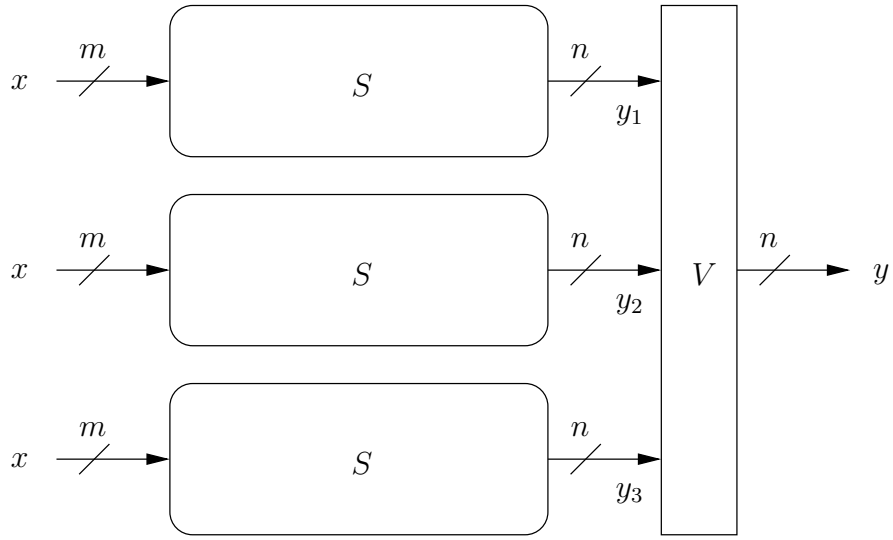

Abbildung 2.1.: Systemarchitektur der dreifach modularen Redundanz [Joh88]

wieder von der korrekten Funktionsweise eines einzelnen Voters ab.

Eine Verbesserung dieser Problematik lässt sich durch selbstprüfende Voter erzielen, wie sie z.B. in [CRM05] vorgestellt wurden. Eine Variante sieht vor, einen internen Fehler dadurch anzuzeigen, dass sich das Ausgangssignal während einer Taktperiode nicht ändert. Entsprechend alterniert das Ausgangssignal während einer Taktperiode, wenn in der Schaltung kein Fehler vorliegt. Bei einer anderen Variante wird dem Voter ein weiterer Ausgang hinzugefügt und das fehlerfreie Ausgangssignal wird vom fehlerhaften Ausgangssignal dadurch unterschieden, dass im fehlerfreien Fall beide Ausgänge am Ende einer Taktperiode denselben Wert tragen. Im Fehlerfall unterschieden sich diese Werte am Ende einer Taktperiode.

Eine weitere Möglichkeit, über die Mehrheit der Ausgaben der drei Schaltungen zu entscheiden, bilden Word-Voter [MM05]. Mit herkömmlichen Votern werden die einzeln Komponenten der drei Ausgabevektoren unabhängig voneinander bearbeitet, ohne dass die Semantik der kodierten Ausgaben berücksichtigt wird. Dieses Vorgehen ist geeignet, wenn sich pro Komponente immer nur eine Ausgabe der drei Schaltungen vom korrekten Ergebnisvektor unterscheidet. D.h., bestimmt die ursprüngliche Schaltung aus ihren Eingaben beispielsweise den korrekten zweistelligen binären Ausgabevektor $y_1 = (00)$ und erzeugen die zusätzlichen Schaltungen z.B. die fehlerhaften Ausgaben $y_2 = (01)$ und $y_3 = (10)$, dann wird ein herkömmlicher Voter die korrekte Ausgabe $y = (00)$ liefern. Berechnen die drei Schaltungen ihre Ausgaben aber auf unterschiedliche Weise, dann ist eine bitweise Mehrheitsentscheidung ungeeignet, da die Bedeutung der Ausgabewörter entscheidend ist. Der bitweise arbeitende Voter bildet im Beispiel also nur zufällig das korrekte Ausgabewort aus drei vollkommen unterschiedlichen Eingabevektoren. Wäre die Ausgabe einer der zusätzlichen Schaltungen, z.B. $y_2 = (11)$, dann wäre auch die Mehrheitsentscheidung des Voters fehlerhaft, ohne dass er diesen Fehler anzeigen würde. Ein

Word-Voter ist genau für diese Problematik konzipiert und wird auf einer zusätzlichen Ausgangsleitung immer dann ein Fehlersignal erzeugen, wenn alle seine Eingabewörter verschieden zueinander sind.
Wenn die Schaltung S sehr komplex ist und viel Fläche einnimmt, kann die Zuverlässigkeit eines TMR-Systems auch dadurch erhöht werden, dass der Voter in einer robusteren Technologie umgesetzt wird. Da der Voter dann im Vergleich zur fehlertoleranten Gesamtschaltung relativ klein ist, würde sich der Mehraufwand für die zuverlässige Technologie durchaus lohnen.
Wie es zuvor im Zusammenhang mit Word-Votern beschrieben wurde, kann die Fehlertoleranz von TMR-Systemen auch dadurch erhöht werden, dass die drei identischen Systeme durch drei funktional gleiche Systeme ersetzt werden. Diese Diversität kann z.B. durch unterschiedliche Gatter-Bibliotheken, Technologien oder Entwurfsstrategien umgesetzt werden. So lässt sich der Einfluss eines Fehlers verringern, da er sich auf unterschiedliche Weise zu den Ausgängen der einzelnen Schaltungen ausbreitet. Allerdings kann es sein, dass zusätzlich unterschiedliche Laufzeiten der drei Schaltungen berücksichtigt werden müssen.
Mit Hilfe von N-MR Systemen, d.h. Systeme, bei denen die ursprüngliche Schaltung S durch N Schaltungen nach dem oben genannten Prinzip ersetzt wird, können auch Fehler toleriert werden, die sich auf der Komponente k gleichzeitig auf die Ausgabe von mehr als einer der N Schaltungen auswirken. Dabei ist $N > 3$ eine ungerade Zahl, damit der Voter immer eine Mehrheit aus seinen Eingaben bestimmen kann. Die Anzahl der Schaltungen, die gleichzeitig auf derselben Komponente ihres Ausgabevektors einen Fehler liefern dürfen, ohne dass die Gesamtausgabe des Voters verfälscht wird, beträgt $(N-1)/2$. Die Kosten für solche Systeme sind jedoch sehr hoch, wodurch ihr Einsatz im Allgemeinen auf äußerst kritische Anwendungen, wie sie z.B. in der Luftfahrt zu finden sind, beschränkt ist.
Um die allgemein sehr hohen Kosten für die Systemvervielfachung zu senken, wird das Verfahren auch auf die fehleranfälligsten Strukturen und Schaltungsteile beschränkt, in denen sich Fehler mit hoher Wahrscheinlichkeit auf das korrekte Verhalten der Schaltung S auswirken. Eine gängige Vorgehensweise besteht in diesem Zusammenhang z.B. darin, nur die Speicherelemente einer Schaltung zu verdreifachen. Zur Identifikation kritischer Bereiche in kombinatorischen Schaltungen werden verschiedene Algorithmen vorgeschlagen [MT03, SS09]. Von Nachteil ist bei dieser partiellen Verdreifachung die Verzögerung des Voters, der hinter jedem verdreifachten Schaltungsteil eingefügt werden muss. Außerdem werden nur die vorab ausgewählten Schaltungsteile gegen Fehler geschützt. Tritt dennoch ein Fehler in einem ungeschützten Schaltungsteil auf, kann sich dieser auch im Fall einer kritischen Berechnung bis zum Ausgang der Gesamtschaltung ausbreiten.

2.2. Fehlererkennung mit Umschaltung

Nach [Sog76] kann Fehlertoleranz auch durch Fehlererkennung und Umschaltung realisiert werden. Die Grundidee besteht darin, die Ausgaben einer gegebenen Schaltung $S = S_1$ im Fehlerfall durch die Ausgaben einer funktional identischen Reserveschaltung

S_2 zu ersetzen. Die Ersetzung übernimmt eine zusätzliche Schaltung B, die durch die Ausgaben f_1 einer Fehlererkennungsschaltung C_1 für S_1 und durch die Ausgaben f_2 einer Fehlererkennungsschaltung C_2 für S_2 gesteuert wird. Die Fehlererkennungsschaltungen sollten selbstprüfend sein, damit bei der Umschaltung auch Fehler in diesen Teilschaltungen berücksichtigt werden. Wie die Fehlererkennung genau umgesetzt wird, ist abhängig vom betrachteten Fehlermodell. In der genannten Quelle wird z.B. die Gruppenparität vorgeschlagen. Die Bestimmung dieser Kodes ist jedoch sehr aufwändig. Weitere Möglichkeiten für den Entwurf von Fehlererkennungsschaltungen werden in [GOSM08] beschrieben.
Der allgemeine Systemaufbau für die Fehlererkennung mit Umschaltung ist beispielhaft für eine Austauschschaltung in Abbildung 2.2 dargestellt. Er kann aber auch auf beliebig viele Austauschschaltungen erweitert werden.

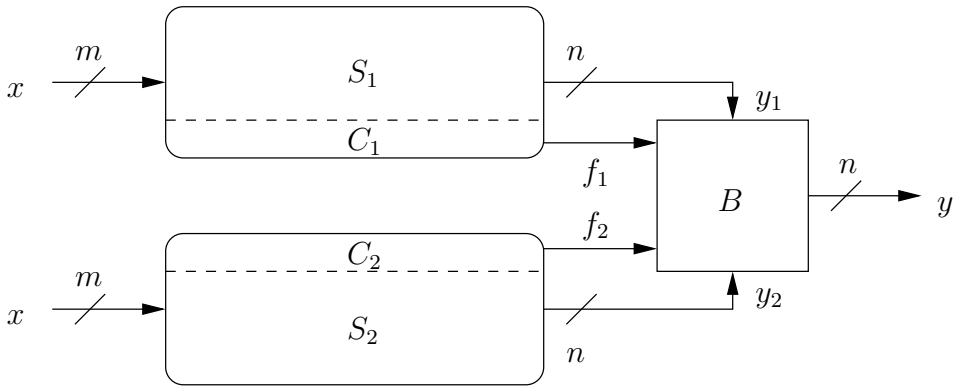

Abbildung 2.2.: Systemarchitektur der Fehlererkennung mit Umschaltung nach [Sog76]

Um das Verfahren im laufenden Betrieb ohne Unterbrechung durchführen zu können, wird die Reserveschaltung S_2 mit ihrer Fehlererkennungsschaltung C_2 parallel zu den Schaltungen S_1 und C_1 betrieben. Die Schaltungen S_1 und S_2 erhalten gleichzeitig dieselben m-stelligen Eingaben $x \in \{0,1\}^m$ und erzeugen parallel dieselben n-stelligen Ausgaben $y_1 = y_2 \in \{0,1\}^n$. Es wird außerdem angenommen, dass die Ausgaben f_1 und f_2 der Fehlererkennungsschaltungen C_1 und C_2 im fehlerfreien Fall den logischen Wert 1 und im Fehlerfall den logischen Wert 0 tragen. Andere Realisierungen wären ebenfalls möglich und werden weiter unten vorgestellt.
Die Schaltung B ist nun so konstruiert, dass sie die Ausgabe des Gesamtsystems y im fehlerfreien Fall durch die beiden Ausgabevektoren y_1 und y_2 bestimmt. Dazu können die Ausgabevektoren y_1 und y_2 komponentenweise z.B. durch ODER-Gatter miteinander verknüpft werden. Im Fehlerfall wird die Ausgabe des Gesamtsystems wiederum nur durch die Ausgabe der fehlerfreien Schaltung bestimmt. Jede Komponente des Ausgabevektors y_1 wird dazu in der Schaltung B vorher noch mit dem Fehlersignal f_1 z.B. durch ein UND-Gatter verbunden und bei y_2 wird die gleiche Verknüpfung über das Fehlersignal f_2 durchgeführt. Auch für die Schaltung B sind weitere Umsetzungsmöglichkeiten, z.B. mit Hilfe eines Multiplexers, der durch die Fehlersignale gesteuert wird, denkbar.

Wird nun in einer der Schaltungen S_1 oder S_2 ein Fehler erkannt, dann wird das entsprechende Fehlersignal den logischen Wert 0 tragen und alle Komponenten des zugehörigen Ausgabevektors werden durch die UND-Verknüpfungen auf den logischen Wert 0 gesetzt. Das Fehlersignal der fehlerfreien Schaltung trägt den logischen Wert 1. Da dieser Wert keinen primären Einfluss auf die Ausgabe der UND-Gatter hat (non-controlling), wird der fehlerfreie Ausgabevektor an den Ausgang der UND-Gatter weitergeleitet.
Der auf den logischen Wert 0 gesetzte fehlerhafte Ausgabevektor hat bei der Bestimmung der Gesamtausgabe durch die nachgeschalteten ODER-Gatter keinen primären Einfluss. Die Ausgabe des Gesamtsystems wird daher nur durch den fehlerfreien Ausgabevektor bestimmt.
Dieses Umschaltprinzip wird in [AM03] auch so realisiert, dass nur die Reserveschaltung S_2 mit einer Fehlererkennung ausgestattet wird. Es wird ebenfalls vorgeschlagen hierfür die Gruppenparität zu verwenden. Das Fehlersignal ist zweistellig und wird durch eine Paritätsprüfung der Reserveschaltung S_2 mit einer Paritätsvorhersageschaltung und den Vergleich der Ausgaben von S_1 und S_2 erzeugt.
Unter der Annahme eines Einzelfehlers gilt dann Folgendes. Zeigt nur die Paritätsprüfung einen Fehler an, dann sind die Ausgaben der Schaltungen S_1 und S_2 gleich und es ist davon auszugehen, dass diese Ausgaben korrekt sind. Für den Fall, dass nur der Vergleich der Ausgabevektoren y_1 und y_2 einen Fehler ergibt, ist die Ausgabe der Schaltung S_2 bezüglich des betrachteten Fehlermodells fehlerfrei, da die Parität für diese Ausgabe korrekt bestimmt werden konnte. Ergibt sich sowohl für die Paritätsprüfung als auch für den Vergleich der Ausgaben von S_1 und S_2 ein Fehler, so ist die Ausgabe von S_1 korrekt und die von S_2 ist inkorrekt. Auf Basis dieser Fehlersignale kann dann die Umschaltung zwischen den Ausgaben von S_1 und S_2 mit Hilfe der Schaltung B erfolgen.
Bei sequentiellen Schaltungen kann diese Technik auch auf die Inhalte der Zustandsregister angewendet werden, um die Zustandsüberführungsfunktion abzusichern. Hierzu wird die Umschaltung auf Basis des aktuellen Zustandes vor der Berechnung des Folgezustandes vorgenommen. Die Berechnung des Fehlersignals kostet allerdings zusätzlich Zeit, wodurch das Gesamtsystem verlangsamt wird.

2.3. Double Modular Redundancy

Als Double Modular Redundancy wird nach [SKJW07] eine Modifikation des TMR-Konzeptes bezeichnet, die statt drei nur zwei funktional identische Schaltungen S_1 und S_2 zur Fehlermaskierung nutzt. Die Ausgänge der zwei Schaltungen werden in die Eingänge eines rückgekoppelten Voters V geführt, der für die Mehrheitsentscheidung zusätzlich seine eigene Ausgabe nutzt. Eine ähnliche Umsetzung dieses Prinzips wird in [MZK09] auch mit Hilfe eines C-Elementes nach [MB59] vorgestellt, das anstelle des rückgekoppelten Voters zum Einsatz kommt. Die Funktionen des rückgekoppelten Voters und des C-Elementes sind äquivalent. Eine entsprechende Wertetabelle ist in Tabelle 2.1 dargestellt.
Der Vorteil des rückgekoppelten Voters besteht jedoch darin, dass er sich auf einfache Weise mit jeder Standardzellenbibliothek implementieren lässt.

Tabelle 2.1.: Wertetabelle des C-Elementes bzw. rückgekoppelten Voters

x_1	x_2	y
0	0	0
0	1	y
1	0	y
1	1	1

Der Effekt der Konstruktion ist, dass der aktuelle Ausgabewert des Voters bzw. C-Elementes so lange beibehalten wird, bis sich die Ausgaben der beiden Schaltungen S_1 und S_2 von der aktuellen Ausgabe des Voters oder C-Elementes unterscheiden.
Auf diese Weise lassen sich spezielle transiente Fehler in bestimmten Systemzuständen tolerieren. Besonders günstig ist dieses Prinzip z.B. im Zusammenhang mit Speicherelementen einsetzbar. Ein solcher Systemaufbau ist am Beispiel von zwei Flip-Flops in Abbildung 2.3 sowohl für den rückgekoppelten Voter (links) als auch für das C-Element (rechts) dargestellt.

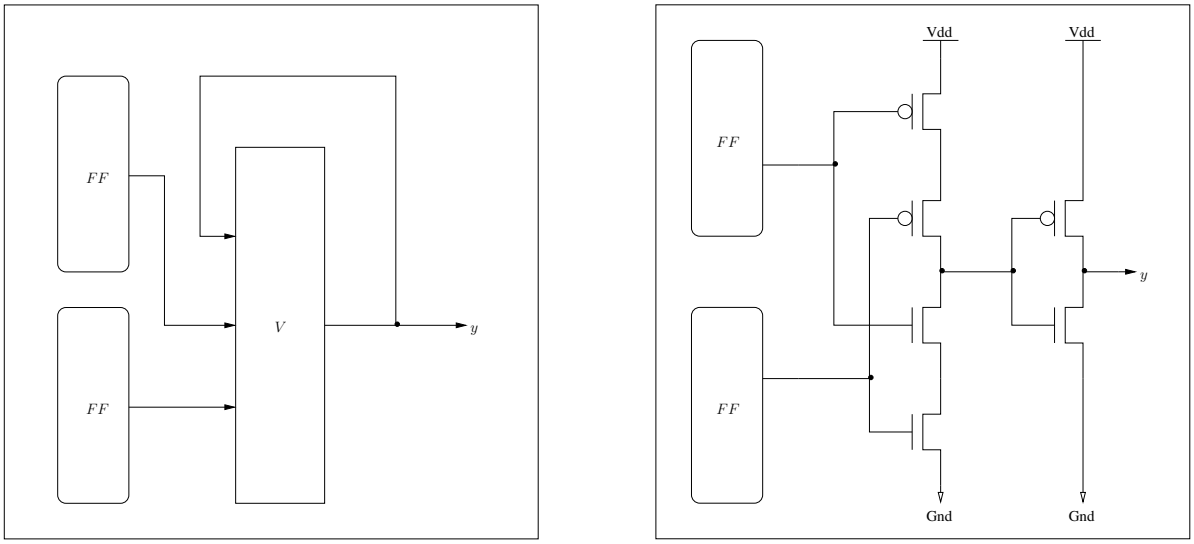

Abbildung 2.3.: Double Modular Redundancy für Flip-Flops [SKJW07, MZK09]

In dem Zeitraum, in dem die Speicherelemente aus Abbildung 2.3 keine neuen Werte aufnehmen, wird jeder transiente Fehler toleriert, der nur eines der Speicherelemente oder die Rückführungsleitung des Voters betrifft. Die aktuelle Ausgabe y bleibt währenddessen erhalten. Auch ein transienter Fehler im Voter bzw. C-Element wird in diesem Zeitraum automatisch wieder ausgeglichen und betrifft die Ausgabe y nur solange, wie der Fehler vorhanden ist. Tritt während des Ladevorgangs in einem der Speicherelemente ein transienter Fehler auf, wird er auch dann noch toleriert, wenn sich der korrekte Wert der Speicherelemente nicht von der aktuellen Ausgabe y und somit auch nicht von dem

Wert auf der Rückführungsleitung unterscheidet. Ebenso wird zu diesem Zeitpunkt ein transienter Fehler im Voter oder C-Element wieder ausgeglichen, da die Speicherelemente den korrekten Wert tragen.
Andere Fehler können sich allerdings schon auf die Ausgabe y auswirken. Zudem muss im Vergleich zur nicht fehlertoleranten Variante eines Systems, das nach diesem Prinzip geschützt wird, für jedes Speicherelement eine zusätzliche Verzögerung durch den Voter oder das C-Element berücksichtigt werden. Der Ansatz bietet aber dennoch einen guten Schutz gegen eine große Menge an Fehlern. Außerdem ist der benötigte Flächenaufwand für dieses Verfahren geringer als die Verwendung von drei Speicherelementen und einem Voter.

2.4. Fehlerkorrigierende Kodes

Zur Fehlererkennung- und Korrektur in Speicherblöcken werden vorrangig fehlerkorrigierende Kodes eingesetzt [CH84]. Das Prinzip dieser Kodes besteht darin, Informationen im Speicher nur in Form spezieller Kodewörter abzulegen, aus denen selbst im Fehlerfall die ursprüngliche Information beim Auslesen eindeutig rekonstruiert werden kann.
Für Speicher eignen sich in diesem Zusammenhang besonders separierbare lineare Blockkodes, da ihre Kodewörter alle dieselbe Länge aufweisen, die einzelnen Stellen in den Kodewörtern sich einheitlich in Informations- und Kontrollbits unterteilen lassen und die Informationsbits bei der Ausgabe nicht extra dekodiert werden müssen.
Damit ein Fehler in einem Kodewort korrigiert werden kann, muss es sich selbst im Fehlerfall noch eindeutig von allen anderen Kodewörtern des Kodes unterscheiden. Das gilt für jedes Kodewort. Die minimale Anzahl der Stellen d in denen sich die fehlerfreien Kodeworte voneinander unterscheiden müssen, kann nach [Dan94] durch die Ungleichung $d \geq 2 \cdot t_{kor} + 1$ beschrieben werden. Die Variable t_{kor} gibt dabei die Anzahl der zu korrigierenden Fehlerstellen an.
Darüber hinaus können Kodes auch so entworfen werden, dass sie gleichzeitig eine bestimmte Anzahl t_{erk} an Fehlern erkennen können. Für d gilt dann ebenfalls nach [Dan94] die Ungleichung $d \geq t_{kor} + t_{erk} + 1$. Diese Variante wird aktuell in den meisten praktischen Anwendungen durch Hamming- [Ham50] oder Hsiao-Kodes [Hsi70] zur 1-Bit-Fehlerkorrektur und 2-Bit-Fehlererkennung realisiert, wobei sich die Dekodierung bei den Hsiao-Kodes etwas effizienter gestaltet.
Die Fehlererkennung- und Korrektur wird bei diesen Kodes durch die Auswertung von Paritäten durchgeführt, die über bestimmte Stellen der Informationsbits gebildet werden. Ein allgemeines Schema einer solchen Fehlerkorrekturschaltung ist in Abbildung 2.4 dargestellt.
Vor dem Speichern wird aus den Informationsbits ein Kodewort erzeugt, welches sich mathematisch gesehen über eine Generatormatrix G bestimmen lässt. Die Generatormatrix G besteht aus so vielen linear unabhängigen Kodewörtern des verwendeten Kodes, wie es Informationsbits gibt. Durch die Multiplikation eines Informationsvektors mit der Generatormatrix G kann dann das entsprechende Kodewort gebildet werden. Für separierbare Blockkodes reicht es aber aus, nur die Kontrollbits zu generieren, da die

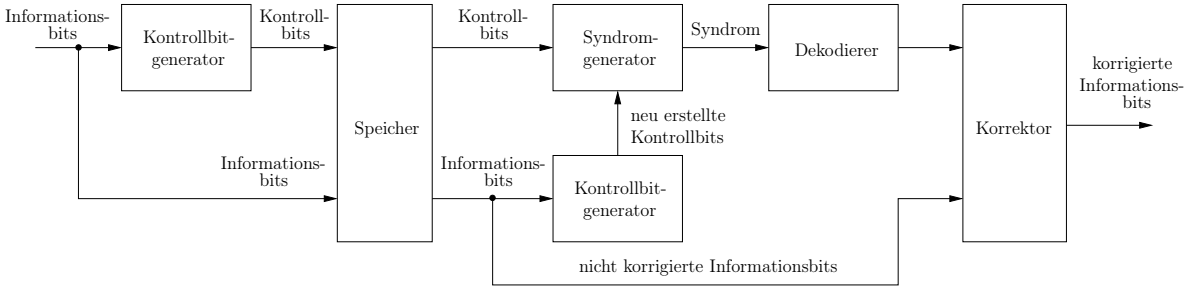

Abbildung 2.4.: Aufbau einer Fehlerkorrekturschaltung für Speicher auf Basis separierbarer Blockkodes nach [Lal01]

Informationsbits unverändert bleiben.
Nachdem das Kodewort wieder aus dem Speicher ausgelesen wurde, werden die Kontrollbits erneut aus den Informationsbits bestimmt. Aus ihnen wird in Verbindung mit den abgespeicherten Kontrollbits ein Syndrom generiert, das angibt ob bzw. auf welcher Position im Kodewort ein Fehler aufgetreten ist. Dieses Syndrom kann auch durch Multiplikation des ausgelesenen Kodewortes mit der Transponierten einer Paritätsprüfmatrix H erzeugt werden, welche sich in einfacher Weise aus der Generatormatrix G bilden lässt.
Zeigt das Syndrom einen speziellen Fehler an, wird ein Signal erzeugt, mit dem dieser Fehler korrigiert werden kann. Entspricht das Syndrom lediglich einem Fehlererkennungssignal, kann der Fehler nicht korrigiert werden. Das kann bei einem 1-Bit fehlerkorrigierenden und 2-Bit fehlererkennenden Kode im Fall eines 2-Bit-Fehlers auftreten.
Theoretisch ist es möglich das beschriebene Prinzip bei den genannten Kodes auch auf zusätzliche Fehler abzubilden. Praktisch eignen sie sich allerdings nicht dazu, da der benötigte Hardware- und Dekodierungsaufwand dafür sehr hoch ist.
Dennoch müssen zunehmend auch Mehrbitfehler von beliebiger Vielfalt in Speicherblöcken berücksichtigt werden. Sie lassen sich bekanntermaßen durch BCH-Kodes [BRC60] und Reed-Solomon-Kodes [HP03] korrigieren. Die Dekodierungsalgorithmen sind für diese Kodes allerdings auch sehr komplex und ihre Implementierung erfordert sowohl einen hohen Hardware- als auch Entwurfsaufwand.

3. Spezifische Fehlertoleranz für kombinatorische Schaltungen

3.1. Grundlegende Systemarchitektur

Die spezifische Fehlertoleranz ist eine Modifikation des TMR-Konzeptes. Im Gegensatz zur Systemverdreifachung kann mit Hilfe dieser neuen Methode eine Schaltung so entworfen werden, dass sie nur für eine Teilmenge aller möglichen Ein- und Ausgabesignale dieselbe Fehlertoleranz bereitstellt, wie es durch ein TMR-System garantiert werden kann. Dieses Prinzip hat den Vorteil, dass der Hardwareaufwand für redundante Systemkomponenten, die zur Realisierung der Fehlertoleranz benötigt werden, exakt auf die tatsächlich vorhandenen Anforderungen einer Anwendung abgestimmt werden kann. In der Praxis ist das von großer Bedeutung, da auf diese Weise ein signifikanter Anteil an Kosten bei der Herstellung fehlertoleranter Systeme eingespart werden kann. Das gilt insbesondere für Systeme, die sowohl kritische als auch unkritische Anwendungen übernehmen, da es in solchen Systemen ausreicht, nur den Teil der Funktionalität gegen Fehler zu schützen, der tatsächlich zur Realisierung kritischer Aufgaben notwendig ist.
Zur praktischen Umsetzung dieser Idee werden einer gegebenen kombinatorischen Schaltung $S = S_1$ mit $m \geq 1$ Eingängen und $n \geq 1$ Ausgängen im einfachsten Fall zwei zusätzliche kombinatorische Schaltungen s_2 und s_3 mit der gleichen Ein- und Ausgabewortbreite hinzugefügt. Wie sich später zeigen wird, ist der Hardwareaufwand für die zusätzlichen Schaltungen s_2 und s_3 geringer als für die Schaltung $S = S_1$. Die Schaltungen S_1, s_2 und s_3 realisieren jeweils die m-stelligen Booleschen Funktionen $S_1(x) = y_1$, $s_2(x) = y_2$ und $s_3(x) = y_3$ mit y_1, y_2, $y_3 \in \{0,1\}^n$. Ähnlich einem TMR-System werden die Ausgänge der drei Schaltungen in die Eingänge eines Voters V geführt und alle Schaltungen S_1, s_2 sowie s_3 erhalten gleichzeitig dieselben Eingaben $x \in X$. Dabei bezeichnet X die Menge aller m-stelligen Booleschen Vektoren $X = \{0,1\}^m$, die als mögliche Eingaben der Form $x = (x_1, \ldots x_m)$ für die drei Schaltungen in Frage kommen. Der entsprechende Systemaufbau ist in Abbildung 3.1 dargestellt.
Damit durch diesen Systemaufbau eine vom Entwerfer beliebig wählbare Teilmenge X_1 aller möglichen Eingaben X in das System fehlertolerant verarbeitet werden kann, müssen die zusätzlichen Schaltungen wie folgt aufgebaut sein. Die Menge X_1 beinhalte alle Eingaben, die in der Spezifikation der Schaltung $S = S_1$ als kritisch eingestuft wurden. Für diese Eingaben $x \in X_1$ soll somit derselbe Grad an Fehlertoleranz bereitgestellt werden, wie es auch durch ein TMR-System garantiert werden kann. Fehler im Voter müssen daher nicht weiter berücksichtigt werden. Für alle Eingaben $x \in X_1$, wobei $X_1 \subseteq X$, sind die zusätzlichen Schaltungen s_2 und s_3 dann so aufzubauen, dass sie dieselben Ausgaben

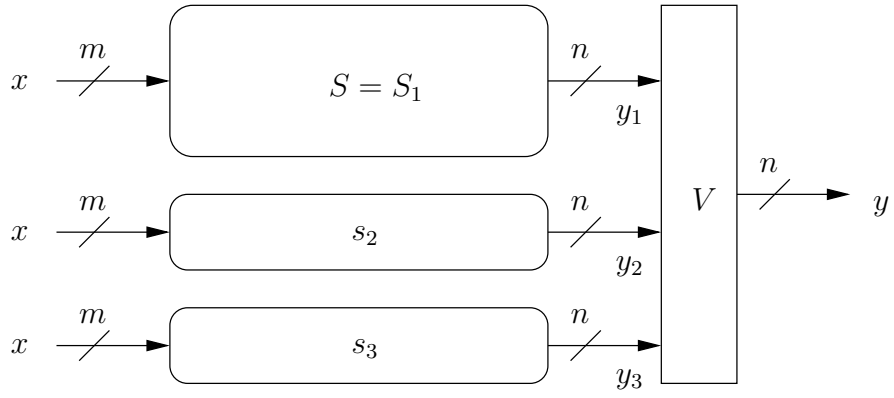

Abbildung 3.1.: Systemarchitektur der spezifischen Fehlertoleranz

liefern, wie die ursprüngliche Schaltung $S = S_1$. Die Funktionen der drei Schaltungen müssen somit die Bedingung

$$S_1(x) = s_2(x) = s_3(x) \quad \text{für} \quad x \in X_1 \tag{3.1}$$

erfüllen, damit bei einer Eingabe $x \in X_1$ jeder Fehler, der nur eine der Schaltungen S_1, s_2 oder s_3 betrifft, toleriert wird.
Bei allen weiteren Eingaben $x \in X \setminus X_1$, die durch den Systemaufbau nicht zwingend fehlertolerant verarbeitet werden müssen, ist mindestens eine der zusätzlichen Schaltungen s_2 oder s_3 so zu entwerfen, dass sie dieselbe Ausgabe liefert, wie die Schaltung S_1. Auf diese Weise wird sichergestellt, dass es zur Erzeugung des korrekten Ausgangssignals der Schaltung S_1 an den Eingängen des Voters V eine Mehrheit gibt. Für die Funktionen der zusätzlichen Schaltungen gilt dann:

$$\Big(S_1(x) = s_2(x)\Big) \quad \text{oder} \quad \Big(S_1(x) = s_3(x)\Big) \quad \text{für} \quad x \in X \setminus X_1. \tag{3.2}$$

Durch die Bedingung aus Formel (3.2) ergibt sich für die spezifische Fehlertoleranz ein entscheidender Vorteil gegenüber der Systemverdreifachung. Da für jeden Eingabevektor $x \in X \setminus X_1$ nur eine der Schaltungen s_2 oder s_3 die Ausgabe des Systems S_1 erzeugen muss, können an diesen Stellen die Ausgaben der jeweils anderen Schaltung zu Optimierungszwecken ausgenutzt werden. Diese Freiheit im Schaltungsentwurf ermöglicht es, den Hardwareaufwand für die zusätzlichen Schaltungen s_2 und s_3 erheblich zu senken. Algorithmen, die dieses Prinzip verfolgen, werden in Kapitel 4 vorgestellt.
Die Bedingung aus Formel (3.2) legt nicht fest, dass die Ausgaben der Funktionen $s_2(x)$ und $s_3(x)$ für $x \in X \setminus X_1$ verschieden sein müssen. Durch den Optimierungsprozess kann es daher auch dazu kommen, dass Eingaben fehlertolerant verarbeitet werden, für die ein solcher Schutz ursprünglich nicht vorgesehen war. Das ist für die Funktionalität des resultierenden Schaltungsaufbaus nicht von Nachteil. In diesem Fall werden lediglich bei möglichst geringem Flächenaufwand mehr Eingaben fehlertolerant verarbeitet werden, als es ursprünglich in der Spezifikation vorgegeben wurde. Die Menge X_1 ist also in

Abhängigkeit des Optimierungsprozesses potentiell etwas größer, was formal dadurch ausgedrückt werden kann, dass $X_1 \subseteq X_{1\_total}$ ist. X_{1_total} ist dabei die Teilmenge aus X, die sämtliche Eingaben enthält, welche nach der Optimierung fehlertolerant verarbeitet werden. Es gilt somit $X_1 \subseteq X_{1\_total} \subseteq X$.
Alle Eingabevektoren, deren zugehörige Ausgabevektoren komponentenweise nach dem Optimierungsprozess eine der folgenden Formeln erfüllen

$$S_1(x) = s_2(x) \quad \text{und} \quad \overline{S}(x) = s_3(x) \tag{3.3}$$

oder

$$\overline{S}_1(x) = s_2(x) \quad \text{und} \quad S(x) = s_3(x) \tag{3.4}$$

werden durch die Systemarchitektur aus Abbildung 3.1, wie durch die Spezifikation gewollt, nicht fehlertolerant verarbeitet. Im Bezug auf den gesamten Ausgabevektor trifft das ebenfalls zu, wenn sich die Ausgaben von s_2 und s_3 in mindestens einer Komponente nach dem Optimierungsprozess unterscheiden. Alle Eingabevektoren, für die durch die Systemarchitektur aus Abbildung 3.1 keine Fehlertoleranz bereitgestellt wird, werden in der Menge X_2 zusammengefasst. Im Allgemeinen gilt: $X_2 \subseteq X \setminus X_{1\_total}$. Die genannten Beziehungen zwischen den einzelnen Teilmengen aller möglichen Eingabevektoren $x \in X$ des Systems sind in Abbildung 3.2 veranschaulicht.

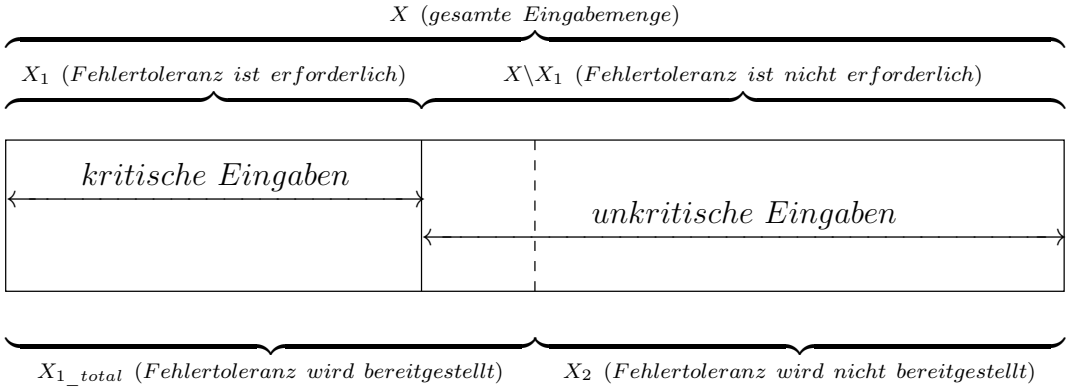

Abbildung 3.2.: Aufteilung der Eingabemenge X

Bei dem Spezialfall, dass die Teilmenge X_2 leer ist und somit $X_1 = X$ ist, entspricht die spezifische Fehlertoleranz einem TMR-Ansatz, da alle möglichen Eingaben fehlertolerant verarbeitet werden sollen. Die zusätzlichen Schaltungen s_2 und s_3 realisieren dann dieselbe Funktion wie die Schaltung S_1 und der Systemaufbau aus Abbildung 3.1 muss durch die Architektur für TMR-Systeme aus Abbildung 2.1 ersetzt werden.
Gleiches gilt, wenn die Schaltungen s_2 und s_3 nicht kleiner als S_1 realisiert werden können. Die Schaltungen s_2 und s_3 nehmen dann maximal die Größe der Schaltung S_1 an, wodurch eine TMR-Realisierung immer die obere Grenze für den benötigten Hardwareaufwand beim Entwurf von Schaltungen nach dem Prinzip der spezifischen Fehlertoleranz darstellt.

3.2. Erweiterungen zur Erhöhung der Zuverlässigkeit

Die im vorangegangenen Abschnitt 3.1 vorgestellte Systemarchitektur garantiert die fehlertolerante Verarbeitung aller als kritisch eingestuften Eingaben $x \in X_1$ einer gegebenen Schaltung $S = S_1$. Für unkritische Eingaben, die der Menge X_2 zugeordnet werden, wird durch den Systemaufbau keine Fehlertoleranz bereitgestellt.
Im Folgenden wird dargestellt, wann sich ein Fehler φ in einer der drei Schaltungen S_1, s_2 oder s_3 bei einer Eingabe $x \in X_2$ auf mindestens einen der n Ausgänge des Gesamtsystems auswirkt. Es wird angenommen, dass der Fehler φ durch die Eingabe $x \in X_2$ in der betroffenen Schaltung aktiviert wird und sich dieser bis zu mindestens einer Komponente des zugehörigen Ausgabevektors der Schaltung ausbreitet. Gilt nach Formel (3.3) oder (3.4) im fehlerfreien Fall für eine Komponente k des Ausgabevektors der drei Schaltungen S_1, s_2 oder s_3

$$S_1(x) = S(x), s_2(x) = S(x) \text{ und } s_3(x) = \overline{S}(x)$$

oder

$$S_1(x) = S(x), s_2(x) = \overline{S}(x) \text{ und } s_3(x) = S(x),$$

dann ist die Ausgabe des Voters auf dieser Komponente $y^k = V(S(x), S(x), \overline{S}(x)) = S(x)$ bzw. $y^k = V(S(x), \overline{S}(x), S(x)) = S(x)$. Wirkt sich ein Fehler φ jedoch auf die Komponente k des Ausgabevektors einer der drei Schaltungen aus, dann führen genau die in Tabelle 3.1 aufgelisteten Fälle zu einer fehlerhaften Ausgabe des Voters V auf der Komponente k und es gilt $y^k = \overline{S}(x)$.

Tabelle 3.1.: Fehler, die sich bei Eingaben aus X_2 am Ausgang des Voters auswirken

Fehler in	Ausgaben der Schaltungen auf der betroffenen Komponente			
	S_1	s_2	s_3	V
S_1	$S_1^\varphi(x) = \overline{S}(x)$	$s_2(x) = S(x)$	$s_3(x) = \overline{S}(x)$	$V(\overline{S}(x), S(x), \overline{S}(x)) = \overline{S}(x)$
	$S_1^\varphi(x) = \overline{S}(x)$	$s_2(x) = \overline{S}(x)$	$s_3(x) = S(x)$	$V(\overline{S}(x), \overline{S}(x), S(x)) = \overline{S}(x)$
s_2	$S_1(x) = S(x)$	$s_2^\varphi(x) = \overline{S}(x)$	$s_3(x) = \overline{S}(x)$	$V(S(x), \overline{S}(x), \overline{S}(x)) = \overline{S}(x)$
s_3	$S_1(x) = S(x)$	$s_2(x) = \overline{S}(x)$	$s_3^\varphi(x) = \overline{S}(x)$	$V(S(x), \overline{S}(x), \overline{S}(x)) = \overline{S}(x)$

Alle anderen Fehler, die pro Ausgabekomponente immer nur eine der drei Schaltungen betreffen, werden durch den Systemaufbau aus Abbildung 3.1 toleriert.
Die Fehler, die in Tabelle 3.1 aufgeführt sind, betreffen lediglich Eingaben der beim Entwurf als unkritisch eingestuften Teilmenge X_2 und müssen daher vom Systemaufbau aus Abbildung 3.1 auch nicht toleriert werden. Im Vergleich zu einer einzelnen, ungeschützten Schaltung S machen die Fehler in den redundanten Schaltungsteilen s_2 und s_3 das Gesamtsystem jedoch etwas unsicherer. Der Grund dafür ist, dass die zusätzlichen Schaltungen bei einer ungeschützten Schaltung S nicht existieren und somit keine Fehler verursachen können. Nur die Fehler in der Schaltung S_1 hätten auch in einer nicht

fehlertoleranten Einzelschaltung S Auswirkungen auf die Ausgabe des Gesamtsystems. Diese Problematik, dass bei einer Eingabe $x \in X_2$ auch Fehler in den redundanten Schaltungen s_2 und s_3 die Ausgabe des Gesamtsystems verfälschen können, wird durch die nachfolgend beschriebenen Erweiterungen gelöst.

3.2.1. Einführung einer Eingabeerkennung

Der Systemaufbau aus Abbildung 3.1 wird in diesem Abschnitt um die Komponente χ und einen Multiplexer MUX erweitert, damit im Fall einer Eingabe $x \in X_2$ der Einfluss der redundanten Systemteile auf die Zuverlässigkeit des Gesamtsystems reduziert wird. Die Fehlertoleranz für Eingaben aus $X \setminus X_2$ bleibt nach wie vor erhalten.
Die Komponente χ ist eine kombinatorische Schaltung mit m Eingängen und einem Ausgang. Sie dient dazu, kritische von unkritischen Eingaben zu unterscheiden und ermöglicht es, die Ausgabe des Gesamtsystems bei allen Eingaben $x \in X_1$ durch den Voter V und bei einer Eingabe $x \in X \setminus X_1$ durch die Schaltung S_1 bestimmen zu lassen. Dazu steuert die Schaltung χ mit ihrer Ausgabe den Multiplexer MUX, der die beiden n Bit breiten Dateneingänge $D0$ und $D1$ besitzt. In Abhängigkeit des Signals an seinem Steuereingang C leitet der Multiplexer die Werte von einem seiner beiden Dateneingänge an seinen n Bit breiten Ausgang OUT wie folgt weiter:

$$MUX(D0, D1, C) = \begin{cases} D0 & \text{für} \quad C = 0 \\ D1 & \text{für} \quad C = 1. \end{cases} \tag{3.5}$$

Die Ausgänge der Schaltung S_1 sind mit dem Dateneingang $D0$ und die Ausgänge des Voters V mit dem Dateneingang $D1$ verbunden. Der Ausgang der Schaltung χ ist am Steuereingang S des Multiplexers angeschlossen. Eine entsprechende Systemarchitektur ist in Abbildung 3.3 dargestellt.

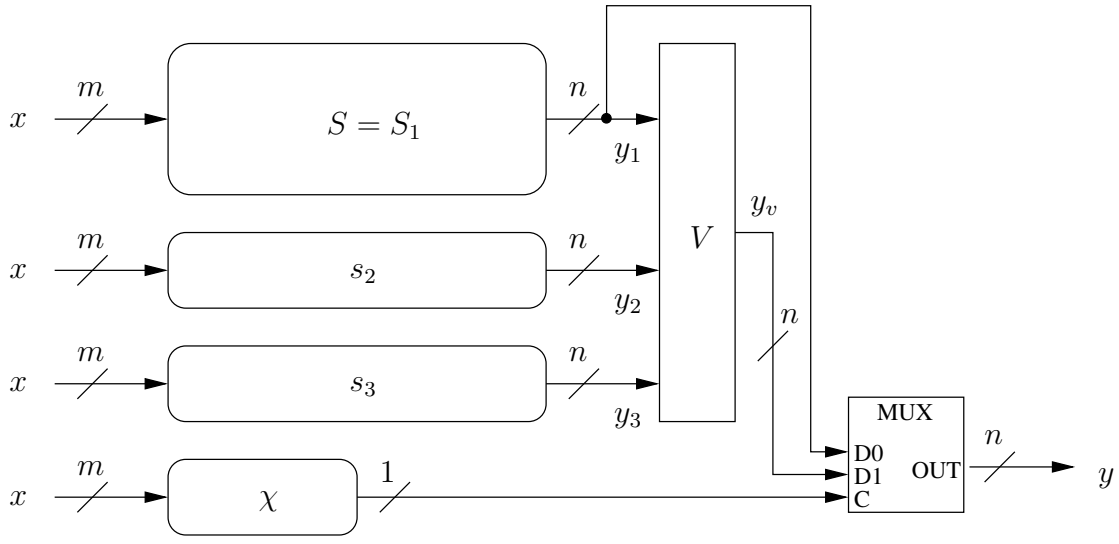

Abbildung 3.3.: Erweiterte Systemarchitektur mit Eingabeerkennung

Die Schaltung χ erhält gleichzeitig mit den Schaltungen S_1, s_2 und s_3 dieselben Eingaben $x \in X$ und realisiert die m-stellige Boolesche Funktion $\chi(x)$, welche folgendermaßen definiert ist:

$$\chi(x) = \begin{cases} 1 & \text{für} \quad x \in X_1 \\ 0 & \text{sonst.} \end{cases} \tag{3.6}$$

Die Funktion $\chi(x)$ ist also die charakteristische Funktion der Eingabeteilmenge X_1, die über den Steuereingang C des Multiplexers MUX festlegt, ob dieser als Ausgabe y des Gesamtsystems die Ausgabe y_v des Voters V oder die Ausgabe y_1 der Schaltung S_1 weiterleiten soll. Beim Entwurf der Schaltung χ sollte beachtet werden, dass der Hardwareaufwand für die Implementierung des in Formel 3.6 dargestellten On-Sets von $\chi(x)$ größer sein kann als für das Off-Set. Wird das Off-Set von $\chi(x)$ verwendet, d.h.

$$\chi(x) = \begin{cases} 1 & \text{für} \quad x \notin X_1 \\ 0 & \text{sonst,} \end{cases} \tag{3.7}$$

dann müssen die Ausgänge des Voters V an den Dateneingang $D0$ und die Ausgänge der Schaltung S_1 an den Dateneingang $D1$ des Multiplexers MUX angeschlossen werden.
Prinzipiell könnte die Funktion $\chi(x)$ auch über die Eingabeteilmenge X_2 realisiert werden. Das ist allerdings für Schaltungen mit großer Eingabewortbreite unpraktisch, da die Teilmenge X_2 erst nach dem Optimierungsprozess exakt bekannt ist und die Bestimmung der einzelnen Vektoren, die nur zu X_2 gehören, dann sehr aufwändig sein kann. Die Menge X_1 ist hingegen von vornherein bekannt.
Wie es nachfolgend gezeigt wird, lassen sich durch die Systemarchitektur aus Abbildung 3.3 im Gegensatz zur spezifischen Fehlertoleranz ohne Eingabeerkennung auch alle Fehler tolerieren, die eine der redundanten Schaltungen s_2, s_3 oder χ betreffen. Das gilt auch für die in Tabelle 3.1 aufgelisteten Fehler der Schaltungen s_2 und s_3, die in der Systemarchitektur ohne Eingabeerkennung zu einer fehlerhaften Gesamtausgabe führen.

1. Bei einer Eingabe $x \in X_1$ gilt für die Ausgaben der Teilschaltungen auf der Komponente k des Ausgabevektors $y = S(x)$ im fehlerfreien Fall $S(x) = S_1(x) = s_2(x) = s_3(x)$ und $\chi(x) = 1$. Der Multiplexer MUX leitet somit die Ausgabe des Voters V als Ausgabe des Gesamtsystems weiter und die in Tabelle 3.2 aufgelisteten Fehler haben keine Auswirkung auf den korrekten Wert der Ausgabekomponente k.

Tabelle 3.2.: Systemverhalten bei einem Einzelfehler, wenn $x \in X_1$

Fehler		Ausgaben der Teilschaltungen					
in	χ	S_1	s_2	s_3	V	MUX	Bemerkung
χ	0	$S(x)$	$S(x)$	$S(x)$	$S(x)$	$S(x)$	$y = S_1(x)$
S_1	1	$\overline{S}(x)$	$S(x)$	$S(x)$	$S(x)$	$S(x)$	$y = V(y_1, y_2, y_3)$
s_2	1	$S(x)$	$\overline{S}(x)$	$S(x)$	$S(x)$	$S(x)$	$y = V(y_1, y_2, y_3)$
s_3	1	$S(x)$	$S(x)$	$\overline{S}(x)$	$S(x)$	$S(x)$	$y = V(y_1, y_2, y_3)$

2. Bei einer Eingabe $x \in X_2$ gilt für die Ausgaben der Teilschaltungen auf der Komponente k des Ausgabevektors $y = S(x)$ im fehlerfreien Fall $S(x) = S_1(x) = s_2(x)$, $\overline{S}(x) = s_3(x)$ und $\chi(x) = 0$. Der Multiplexer MUX leitet somit die Ausgabe der Schaltung S_1 als Ausgabe des Gesamtsystems weiter und die in Tabelle 3.3 aufgelisteten Fehler haben keine Auswirkung auf den korrekten Wert der Ausgabekomponente k.

Tabelle 3.3.: Systemverhalten bei einem Einzelfehler, wenn $x \in X_2$ und $\overline{S}(x) = s_3(x)$

Fehler		Ausgaben der Teilschaltungen					
in	χ	S_1	s_2	s_3	V	MUX	Bemerkung
χ	1	$S(x)$	$S(x)$	$\overline{S}(x)$	$S(x)$	$S(x)$	$y = V(y_1, y_2, y_3)$
s_2	0	$S(x)$	$\overline{S}(x)$	$\overline{S}(x)$	$\overline{S}(x)$	$S(x)$	$y = S_1(x)$
s_3	0	$S(x)$	$S(x)$	$S(x)$	$S(x)$	$S(x)$	$y = S_1(x)$

3. Bei einer Eingabe $x \in X_2$ gilt für die Ausgaben der Teilschaltungen auf der Komponente k des Ausgabevektors $y = S(x)$ im fehlerfreien Fall $S(x) = S_1(x) = s_3(x)$, $\overline{S}(x) = s_2(x)$ und $\chi(x) = 0$. Der Multiplexer MUX leitet somit die Ausgabe der Schaltung S_1 als Ausgabe des Gesamtsystems weiter und die in Tabelle 3.4 aufgelisteten Fehler haben keine Auswirkung auf den korrekten Wert der Ausgabekomponente k.

Tabelle 3.4.: Systemverhalten bei einem Einzelfehler, wenn $x \in X_2$ und $\overline{S}(x) = s_2(x)$

Fehler		Ausgaben der Teilschaltungen					
in	χ	S_1	s_2	s_3	V	MUX	Bemerkung
χ	1	$S(x)$	$\overline{S}(x)$	$S(x)$	$S(x)$	$S(x)$	$y = V(y_1, y_2, y_3)$
s_2	0	$S(x)$	$S(x)$	$S(x)$	$S(x)$	$S(x)$	$y = S_1(x)$
s_3	0	$S(x)$	$\overline{S}(x)$	$\overline{S}(x)$	$\overline{S}(x)$	$S(x)$	$y = S_1(x)$

Ein Einzelfehler wirkt sich also nur dann auf die Ausgabe des Gesamtsystems aus, wenn er in der Schaltung S_1 bei einer Eingabe aus X_2 auftritt. In diesem Fall gilt $\chi(x) = 0$ und der Multiplexer MUX leitet die Ausgabe der Schaltung S_1 als Ausgabe des Gesamtsystems weiter. Alle anderen Einzelfehler, die nicht den Voter oder den Multiplexer betreffen, werden durch die Systemarchitektur toleriert. Fehler im Voter oder Multiplexer könnten z.B. durch Verdreifachung dieser Teilschaltungen behandelt werden.

Für zusätzliche Fehler, die mehr als eine Schaltung betreffen, gilt, dass die Ausgabe des Gesamtsystems genau dann korrekt ist, wenn trotz der Fehler die Ausgabe des Voters V gleich $S(x)$ und $\chi(x) = 1$ ist oder wenn die Ausgabe von S_1 gleich $S(x)$ und $\chi(x) = 0$ ist.

3.2.2. Abbildung des Grundkonzeptes auf N-MR Systeme

In diesem Abschnitt wird gezeigt, wie das grundlegende Konzept der spezifischen Fehlertoleranz auf N-MR Systeme übertragen werden kann. Auf diese Weise werden alle möglichen Eingaben $x \in X$, die an den Eingängen des Systems angelegt werden können, fehlertolerant verarbeitet und für bestimmte Eingaben $x \in X_1 \subseteq X$, die in der Spezifikation des Systems als besonders kritisch eingestuft wurden, kann zusätzlich ein besonders hoher Schutz garantiert werden.
Im einfachsten Fall wird eine gegebene, nicht fehlertolerante kombinatorische Schaltung S nach dem TMR-Prinzip durch die drei funktional identischen Schaltungen S_1, S_2 und S_3 ersetzt. Für $x \in X$ gilt dann $S(x) = S_1(x) = S_2(x) = S_3(x)$ und jede Eingabe wird durch diesen Systemaufbau fehlertolerant verarbeitet. Um für die speziellen Eingaben $x \in X_1$ einen besonders hohen Grad an Fehlertoleranz zu gewährleisten, werden der nun gegebenen TMR-Schaltung die zwei Schaltungen s_4 und s_5 hinzugefügt. Ihre Ein- und Ausgabewortbreite entspricht der Schaltung S und die Ausgänge dieser zusätzlichen Schaltungen werden ebenfalls in den Voter V geführt. Wie es auch bei herkömmlichen N-MR Systemen der Fall ist, erhalten alle Schaltungen gleichzeitig dieselben Eingaben. Der Voter V bestimmt aus den fünf Ausgabevektoren der Schaltungen S_1, S_2, S_3, s_4 und s_5 ein Mehrheitssignal und leitet dieses an seinen Ausgang weiter. Eine entsprechende Systemarchitektur ist in Abbildung 3.4 dargestellt.

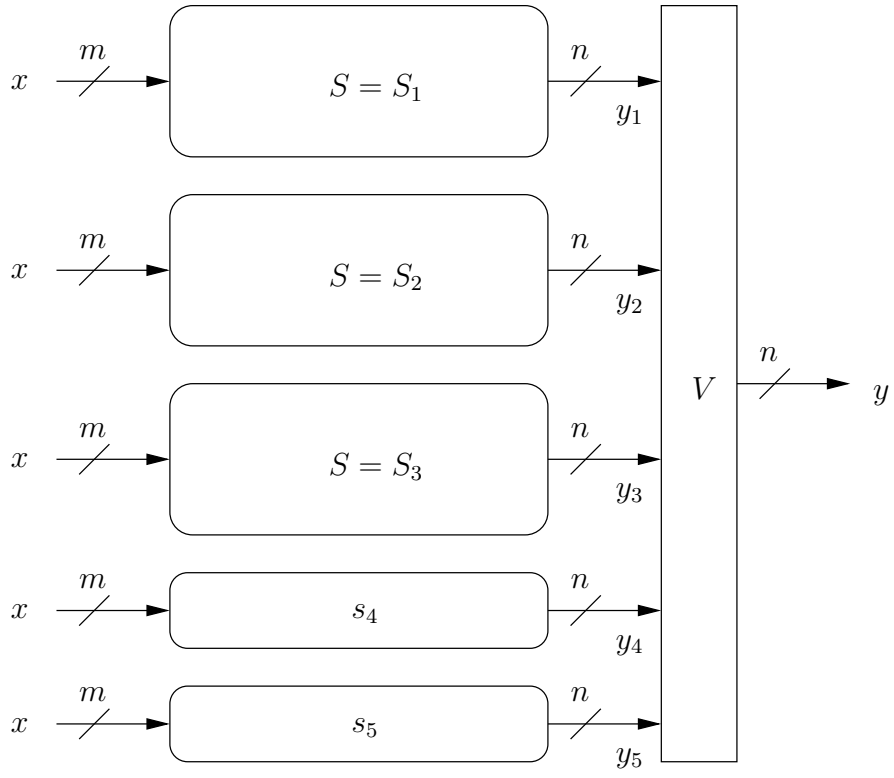

Abbildung 3.4.: Beispiel einer Systemarchitektur mit besonders hoher Fehlertoleranz

Während die Funktionen der Schaltungen S_1, S_2 und S_3 für jede beliebige Eingabe $x \in X$ identisch sind, gilt für die Funktionen der zusätzlichen Schaltungen s_4 und s_5 gemäß den Formeln 3.1 und 3.2 auf jeder Komponente des Ausgabevektors

$$S(x) = s_4(x) = s_5(x) \quad \text{für} \quad x \in X_1$$

und

$$\Big(S(x) = s_4(x)\Big) \quad \text{oder} \quad \Big(S(x) = s_5(x)\Big) \quad \text{für} \quad x \in X \setminus X_1.$$

Die Ausgabewerte der Schaltungen s_4 und s_5, die aufgrund der vorangegangenen Formeln unspezifiziert bleiben, können auch bei diesem Schaltungsaufbau wieder zu Optimierungszwecken ausgenutzt werden. Die Teilmenge der Eingaben aus $X \setminus X_1$, für die dann nach dem Optimierungsprozess entweder

$$S(x) = s_4(x) \quad \text{und} \quad \overline{S}(x) = s_5(x)$$

oder

$$\overline{S}(x) = s_4(x) \quad \text{und} \quad S(x) = s_5(x)$$

gilt, wird wieder als X_2 bezeichnet. Bei einer Eingabe $x \in X_2$ wird durch die Systemarchitektur aus Abbildung 3.4 jeder Fehler toleriert, der sich auf die k-te Komponente des Ausgabevektors einer der Schaltungen S_1, S_2, S_3, s_4 oder s_5 auswirkt und dazu führt, dass auf dieser Komponente neben einer der Schaltungen s_4 oder s_5 zusätzlich die Ausgabe einer der übrigen Schaltungen gleich $\overline{S}(x)$ ist.
Für alle übrigen Eingaben $x \in X_{1\_total} = X \setminus X_2 \supseteq X_1$ gilt, dass alle Fehler toleriert werden, die sich auf die k-te Komponente des Ausgabevektors von zwei der fünf Schaltungen S_1, S_2, S_3, s_4 oder s_5 auswirken, sodass zwei Schaltungen auf dieser Komponente die Ausgabe $\overline{S}(x)$ liefern.
Weitere Fehler lassen sich tolerieren, indem der Systemaufbau aus Abbildung 3.4 um eine gerade Anzahl an Schaltungen mit der Funktionalität der ursprünglichen Schaltung S erweitert wird. Allgemein gilt, ist die Anzahl aller Schaltungen gleich N, dann werden alle Fehler toleriert, die dazu führen, dass maximal $(N-1)/2$ Schaltungen auf der k-ten Komponente ihres Ausgabevektors den Wert $\overline{S}(x)$ tragen.

4. Synthese und experimentelle Ergebnisse

In diesem Kapitel werden heuristische Verfahren vorgestellt, mit denen die redundanten Schaltungsteile s_2 und s_3 für die Systemarchitektur der spezifischen Fehlertoleranz aus Abbildung 3.1 erzeugt werden können. Die Verfahren sind speziell an verschiedene Anforderungen im Entwurfsprozess, wie etwa die Art der Schaltungsbeschreibung oder Rechenaufwand zur Erzeugung der zusätzlichen Schaltungsteile, angepasst.

4.1. Partiell definierte Boolesche Funktionen

In der Definition der spezifischen Fehlertoleranz wird über die Formel 3.2 die Bedingung formuliert, dass bei einer Eingabe aus der unkritischen Teilmenge $X \setminus X_1$ nur mindestens eine der beiden zusätzlichen Schaltungen s_2 oder s_3 komponentenweise dieselbe Ausgabe liefern muss wie die ursprüngliche Schaltung $S = S_1$. Die übrigen unspezifizierten Ausgaben von s_2 und s_3 können beliebige Werte annehmen. Wie es nachfolgend gezeigt wird, kann diese Freiheit ideal zur Reduzierung des Flächenaufwandes der Schaltungen s_2 und s_3 ausgenutzt werden.
Es wird davon ausgegangen, dass $S = S_1$ eine kombinatorische Schaltung mit m Eingängen und einem einzelnen Ausgang ist, welche die m-stellige Boolesche Funktion $y = S_1(x_1, \ldots, x_m) = S_1(x)$ realisiert. Diese vereinfachte Annahme dient der Übersichtlichkeit bei der weiteren Beschreibung und kann problemlos auf Schaltungen mit beliebig breitem Ausgabevektor abgebildet werden, da jede Ausgabekomponente einer solchen Schaltung durch eine separate m-stellige Boolesche Funktion mit einem 1 Bit breiten Ausgabevektor repräsentiert werden kann.
Zunächst legt der Entwerfer fest, welche Eingaben der Schaltung $S = S_1$ fehlertolerant verarbeitet werden sollen. Diese Eingaben werden der kritischen Eingabemenge X_1 zugeordnet und die Boolesche Funktion der Schaltung s_2 wird wie folgt beschrieben

$$s_2(x) = \begin{cases} S_1(x) & \text{für} & x \in X_1 \\ - & \text{don't-care} & \text{sonst.} \end{cases} \tag{4.1}$$

Im Anschluss wird für die nun partiell definierte Boolesche Funktion $s_2(x)$ eine vollständig definierte Funktion $s_2(x)_{opt}$ gesucht, welche mit möglichst wenigen Primimplikanten die bereits spezifizierten Ausgabewerte der Funktion $s_2(x)$ überdeckt. Der Grund dafür ist, dass diese minimierte Funktion nach der Synthese weniger Schaltelemente benötigt als eine Funktion, die durch viele Primimplikanten beschrieben wird.

Zur Minimierung können, je nach Komplexität der Funktion $s_2(x)$, exakte oder heuristische Verfahren verwendet werden. Mit Hilfe exakter Verfahren, wie dem Quine-McCluskey-Algorithmus [McC56], wird $s_2(x)_{opt}$ als Funktion mit einer minimalen Anzahl an Primimplikanten bestimmt. Die Berechnung dieser minimalen Funktion ist jedoch sehr aufwändig und für Schaltungen mit großer Eingabewortbreite nicht praktikabel.
Eine Lösung für diese Problematik liefern heuristische Methoden, durch die die minimale Repräsentation der Funktion $s_2(x)$ gut angenähert werden kann. Ein solches Verfahren ist z.B. der Espresso-Algorithmus [BSVMH84], der auch in diesem Abschnitt zur Optimierung größerer Schaltungen verwendet wird. Er eignet sich zur Berechnung von Problemen mit einer Eingabewortbreite von bis zu 20 Bit und findet auch Lösungen für Funktionen, bei denen exakte Methoden bereits versagen. Heuristiken für beliebig große Schaltungen werden in den nachfolgenden Abschnitten dieses Kapitels erläutert.
Nachdem die Minimierung der Funktion $s_2(x)$ durchgeführt wurde, kann die Funktion der Schaltung s_3 wie folgt beschrieben werden

$$s_3(x) = \begin{cases} S_1(x) & \text{für} & x \in X_1 \\ S_1(x) & \text{für} & S_1(x) \neq s_2(x)_{opt} \\ - & \text{don't-care} & \text{sonst.} \end{cases} \tag{4.2}$$

Auch die partiell definierte Boolesche Funktion $s_3(x)$ wird anschließend minimiert und die resultierende Funktion $s_3(x)_{opt}$ ist vollständig definiert.
Da die Formulierung der Funktionen von s_2 und s_3 nicht explizit vorgenommen wird, sondern im Fall von $s_2(x)$ über $S_1(x)$ und im Fall von $s_3(x)$ über $S_1(x)$ und $s_2(x)$, kann es beim Entwurfsprozess zu Problemen kommen. Einerseits ist es nicht mit jeder Hardwarebeschreibungssprache möglich, Schaltungen nach den Formeln 4.1 und 4.2 implizit über das Verhalten anderer Schaltungen zu definieren und andererseits ist auch nicht jedes Synthesewerkzeug in der Lage, eine solche Beschreibung optimal zu verarbeiten. Es werden daher nachfolgend Möglichkeiten zur expliziten Bestimmung der zusätzlichen Schaltungen s_2 und s_3 vorgestellt.

4.1.1. Präzises Verfahren mit Hilfe von Wertetabellen

Wenn die Eingabewortbreite m der Schaltung $S = S_1$ klein ist, können die Formeln 4.1 und 4.2 über Wertetabellen berechnet und realisiert werden. Der notwendige Rechenaufwand ist bis zu einer Wortbreite von $m \leq 20$ Bit noch akzeptabel und auch gängige Hardwarebeschreibungssprachen bieten die Möglichkeit einer Implementierung von kombinatorischer Logik mit Hilfe von Wertetabellen.
Das On-Set und das Off-Set der Schaltung s_2 wird dazu für alle Eingaben aus X_1 durch Simulation der Schaltung S_1 bestimmt. Das Don't-Care-Set ergibt sich entsprechend aus den übrigen Werten $X \setminus X_1$. Die partiell definierte Schaltung s_2 wird dann durch Anwendung eines Minimierungsverfahrens bezüglich ihrer Fläche optimiert und die zugehörige Funktion $s_2(x)_{opt}$ kann explizit in Form einer Wertetabelle angegeben werden.
Sobald die explizite Beschreibung der Schaltung s_2 vorliegt, kann die Erzeugung der Funktion $s_3(x)$ vorgenommen werden. Da die Schaltungen S_1, s_2 und s_3 nach der Voraussetzung aus Formel 3.1 für $x \in X_1$ dieselben Ausgaben liefern müssen, können beim

Entwurf der Schaltung s_3 die zuvor für s_2 bestimmten Werte, die durch die Simulation von S_1 gewonnen wurden, übernommen werden.
Für die weitere Bestimmung der Funktion $s_3(x)$ sind nun noch die Unterschiede zwischen den Ein- und Ausgabepaaren der Schaltungen S_1 und s_2 von Interesse. Um nicht alle Elemente der beiden Mengen miteinander vergleichen zu müssen, kann die symmetrische Differenz D zwischen den Eingaben der On-Sets X_{on} oder der Off-Sets X_{off} beider Funktionen bestimmt werden. Für das On-Set ist die symmetrische Differenz wie folgt definiert:

$$D_{on} := (S_1^{X_{on}} \setminus s_2^{X_{on}}) \cup (s_2^{X_{on}} \setminus S_1^{X_{on}})$$

Die Ergebnismenge enthält in diesem Fall also nur Eingaben, die zum On-Set einer der beiden Funktionen gehören und nicht dem On-Set beider Funktionen zugeordnet werden können.
Für die sich ergebende Menge an Eingaben unterscheiden sich die Ausgaben der Funktionen $S_1(x)$ und $s_2(x)$, denn wenn eine Eingabe x dem On-Set von nur einer Funktion zugeordnet werden kann, dann muss sie entsprechend im Off-Set der anderen Funktion enthalten sein. Eine Bestimmung über das Off-Set ist äquivalent.
Den aus der symmetrischen Differenz bestimmten Eingaben werden die Ausgaben der Funktion S_1 zugeordnet und in die Beschreibung von s_3 aufgenommen, damit für alle $x \in X \setminus X_1$ die Voraussetzung aus Formel 3.2 erfüllt ist. Die noch nicht spezifizierten Ausgaben der Funktion $s_3(x)$ werden, wie in Formel 4.2 beschrieben, auf $-$ (don't-care) gesetzt. Mit einer abschließenden Minimierung der Funktion ist $s_3(x)$ vollständig definiert und für das Gesamtsystem sind die notwendigen Voraussetzungen aus den Formeln 3.1 und 3.2 erfüllt.

Beispiel

In diesem Abschnitt wird die vorgestellte Heuristik anhand eines Beispiels erklärt. Es wird angenommen, dass die Schaltung S_1 die Boolesche Funktion

$$S_1(x) = (x_1 \oplus x_3) \vee (x_1 \wedge x_2 \wedge x_3)$$

implementiert und dass die Teilmenge X_1 der Menge $X = \{0,1\}^3$ aus den folgenden Elementen besteht

$$X_1 = \{000, 001, 011\}.$$

Die Wertetabelle von $S_1(x)$ ist in Tabelle 4.1 gegeben. Die zu X_1 gehörenden Eingaben $000, 001$ und 011 sind in der mit X_1 bezeichneten Spalte markiert. Die Boolesche Funktion $s_2(x)$ muss für diese Werte das gleiche Ergebnis liefern wie $S_1(x)$. Die entsprechenden Ausgaben von s_2 sind in der Spalte $s_2(x)$ abgebildet. Alle anderen Werte in dieser Spalte sind beliebig wählbar und werden daher auf $-$ (don't-care) gesetzt. Als nächstes wird die partiell definierte Funktion $s_2(x)$ minimiert. Der Vollständigkeit halber ist eine Beispielberechnung im Anhang A zu finden, die auch die Komplexität des

Quine-McCluskey-Algorithmuses widerspiegelt. Als Resultat ergibt sich die Funktion $s_2(x)_{opt} = x_3$, welche in Spalte $s_2(x)_{opt}$ dargestellt ist.

Tabelle 4.1.: Beispiel für die Erzeugung von s_2

x_1	x_2	x_3	$S_1(x)$	X_1	$s_2(x)$	$s_2(x)_{opt}$
0	0	0	0	●	0	0
0	0	1	1	●	1	1
0	1	0	0		-	0
0	1	1	1	●	1	1
1	0	0	1		-	0
1	0	1	0		-	1
1	1	0	1		-	0
1	1	1	1		-	1

Nun kann die Funktion $s_3(x)$ erstellt werden. Zur Verdeutlichung der dabei vorzunehmenden Schritte dient Tabelle 4.2.

Tabelle 4.2.: Beispiel für die Erzeugung von s_3

x	$S_1(x)$	X_1	$s_2(x)_{opt}$	$s_3(x)$	X_2	$s_3(x)_{opt}$
000	0	●	0	0		0
001	1	●	1	1		1
010	0	$\xrightarrow{opt}$ ●	0	-		0
011	1	●	1	1		1
100	1		0	1	●	1
101	0		1	0	●	0
110	1		0	1	●	1
111	1		1	-	$\xrightarrow{opt}$ ●	0

Sowohl für $x \in X_1$ als auch für $s_2(x)_{opt} \neq S_1(x)$, d.h. für die Eingabewerte

$$x \in \{000, 001, 011, 100, 101, 110\},$$

muss die Ausgabe der Funktion $s_3(x)$ gleich der Ausgabe von $S_1(x)$ sein. Für die übrigen Eingabewerte $\{010, 111\}$ kann die Funktion beliebige, d.h. don't-care Werte annehmen. Diese Werte sind in der Spalte $s_3(x)$ von Tabelle 4.2 dargestellt. Nach der Minimierung von $s_3(x)$ ergibt sich somit

$$s_3(x)_{opt} = x_1 \oplus x_3.$$

Die zugehörigen Werte dieser Funktion sind in der Spalte $s_3(x)_{opt}$ von Tabelle 4.2 aufgelistet.

Das System besitzt nun für die Eingabewerte **000**, **001**, 010 und **011** das gleiche Maß an Fehlertoleranz, wie es auch durch ein TMR-System garantiert werden kann, da das Verhalten aller drei Schaltungen S_1, s_2 und s_3 für diese Eingaben identisch. Die Eingaben aus X_1 sind hervorgehoben.
Neben den Eingaben aus X_1 wird zusätzlich eine weitere Eingabe fehlertolerant verarbeitet, denn 010 war ursprünglich Element der Menge $X \setminus X_1 = \{010, 100, 101, 110, 111\}$, für die die Fehlertoleranz nicht zwingend gefordert wurde. An dieser Stelle zeigt sich somit der in Abschnitt 3.1 beschriebene Effekt, dass die Menge aller Eingabevektoren X_{1_total}, die nach der Minimierung fehlertolerant verarbeitet werden, größer sein kann als die Menge X_1, für die die Fehlertoleranz ursprünglich nach einer entsprechenden Spezifikation gefordert wurde. Der Grund dafür liegt darin, dass bei der Minimierung von $s_3(x)$ potentiell jeder don't-care Wert auf den gleichen Wert wie $S_1(x)$ und $s_2(x)_{opt}$ gesetzt werden kann. Es gilt also $X_1 \subseteq X_{1\_total}$ und für alle $x \in X_{1\_total}$ ergibt sich der gleiche Grad an Fehlertoleranz wie für ein TMR-System.
Die Ausgaben von $s_2(x)$ und $s_3(x)$ unterscheiden sich somit nach der Optimierung nur noch für die Eingabeteilmenge $X_2 = \{100, 101, 110, 111\}$, deren entsprechende Werte in der Spalte X_2 markiert sind. Alle diese Eingaben aus X_2 werden von dem Systemaufbau nicht fehlertolerant verarbeitet, da nach den Formeln 3.3 und 3.4 entweder

$$S_1(x) = s_2(x) \quad \text{oder} \quad S_1(x) = s_3(x) \quad \text{mit} \quad s_2(x) \neq s_3(x)$$

erfüllt ist und somit mindestens eine der zusätzlichen Schaltungen s_2 oder s_3 eine andere Ausgabe liefert als die Schaltung S_1. Es gilt nach der Minimierung $X_2 = X \setminus X_{1\_total}$ und das Gesamtsystem verarbeitet in jedem Fall alle Eingaben $X \setminus X_2$ fehlertolerant.

Experimentelle Ergebnisse des Beispiels

Die aus dem Beispiel resultierenden Schaltungen können durch die Formeln

$$S_1(x) = (x_1 \oplus x_3) \vee (x_1 \wedge x_2 \wedge x_3),$$
$$s_2(x) = x_3,$$
$$s_3(x) = x_1 \oplus x_3$$

beschrieben werden. Sie wurden für eine Gegenüberstellung mit einem TMR-Ansatz in einer 0.13 μm Standardzellenbibliothek implementiert. Um eine einheitlich Basis für diesen Vergleich zu erhalten, wurde auch die Schaltung S_1 mit einem Synthesewerkzeug hinsichtlich ihres Flächenaufwandes optimiert. Das Ergebnis ist in Abbildung 4.1 dargestellt.
Es ist zu sehen, dass die Schaltung s_2 lediglich durch eine Leitung und die Schaltung s_3 durch nur ein XOR-Gatter implementiert werden konnte. Die kleinste Realisierung der Schaltung S_1 benötigte wiederum drei Standardzellen.
Die Größen der einzelnen Schaltungen und die des Gesamtsystems sind in Tabelle 4.3 angegeben. Die Werte sind gerundet und setzen sich aus den Flächen der jeweils benötigten Standardzellen in μm^2 zusammen. Der Aufwand für Leitungen und die Platzierung wurde nicht berücksichtigt.

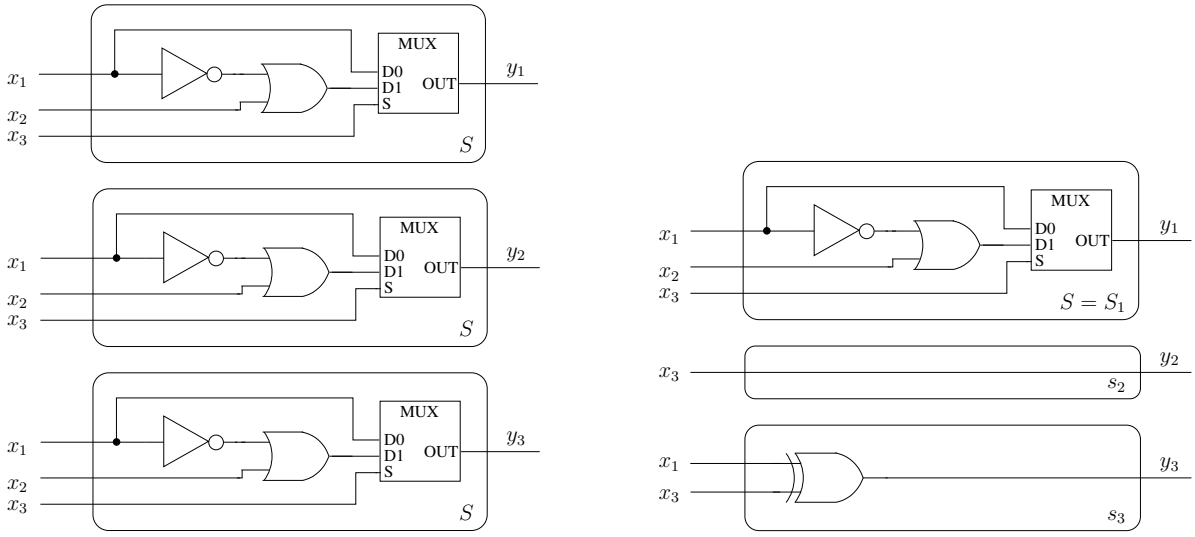

Abbildung 4.1.: Vergleich zwischen TMR (links) und spezifischer Fehlertoleranz (rechts)

Tabelle 4.3.: Flächenaufwand des Beispiels

	TMR	SFT	Reduzierung
Schaltung 1	30 μm^2	30 μm^2	0%
Schaltung 2	30 μm^2	0 μm^2	100%
Schaltung 3	30 μm^2	13 μm^2	56.7%
gesamt	90 μm^2	43 μm^2	52.2%

Die ursprüngliche Schaltung S hat eine Größe von 30 μm^2. Eine TMR-Realisierung von S würde also im Bezug auf die Fläche einen Mehraufwand von 60 μm^2 benötigen. Der Mehraufwand der Implementierung über die spezifische Fehlertoleranz (SFT) beträgt dagegen nur 13 μm^2, was weniger als die Größe von anderthalb Teilsystemen eines TMR-Ansatzes entspricht. Die Flächeneinsparung der spezifischen Fehlertoleranz beträgt gegenüber der TMR-Implementierung insgesamt 52.2%, während noch immer 50% aller möglichen Eingaben geschützt sind.
Das Verfahren wird nun anhand der kleinen Beispielschaltung hinsichtlich der Varianz der zu schützenden Eingabemenge X_1 untersucht. Da die Schaltung $S = S_1$ durch lediglich $k = 8$ Eingabewerte beschrieben wird, ist es möglich, für jede der $\binom{8}{k}$ möglichen Teilmengen X_1 aus X, wobei $k = 0, \ldots, 8$, die zugehörigen

$$\sum_{k=0}^{8} \binom{8}{k} = 256$$

fehlertoleranten Schaltungsanordnungen zu bestimmen. Für alle diese Teilmengen wurde der notwendige Flächenaufwand für die Schaltungen S_1, s_2 und s_3 nach der zuvor

beschriebenen Methode mit derselben 0.13 μm Standardzellenbibliothek bestimmt. Die Schaltungsoptimierung erfolgte über den Espresso-Algorithmus. Die Ergebnisse sind für $k = 0, \ldots, 8$ in Tabelle 4.4 zusammengefasst. Für jedes k wurde dabei im ersten Abschnitt von Tabelle 4.4 der minimale, maximale und durchschnittliche Flächenaufwand jeder Teilschaltung aufgelistet. Die Zeile "gesamt" enthält die jeweils durchschnittliche Größe des Gesamtsystems und die durchschnittliche Flächeneinsparung im Vergleich zu einem TMR-Ansatz ist in der letzten Zeile angegeben.

Tabelle 4.4.: Varianz im Flächenaufwand der Beispielschaltung

		Anzahl geschützter Eingaben								
		8	7	6	5	4	3	2	1	0
System	Fläche	Fläche in μm^2								
S_1	min	30	30	30	30	30	30	30	30	30
	max	30	30	30	30	30	30	30	30	30
	⌀	30	30	30	30	30	30	30	30	30
s_2	min	30	8	8	0	0	0	0	0	0
	max	30	30	30	30	22	13	5	0	0
	⌀	30	23	18	13	9	4	1	0	0
s_3	min	30	30	16	13	8	8	8	8	0
	max	30	30	30	30	30	30	30	13	0
	⌀	30	30	28	26	24	23	19	11	0
gesamt	⌀	90	83	76	69	63	57	50	41	30
		Reduzierung gegenüber TMR in %								
gesamt	⌀	0	8	16	23	30	37	44	54	67

Die Durchschnittswerte der benötigten Flächen von S_1, s_2, s_3 sowie die des gesamten fehlertoleranten Systems (ohne den Voter) sind in Abbildung 4.2 dargestellt.
Die Ergebnisse zeigen, dass die Gesamtfläche fast linear um etwa 7.5% mit der Größe von X_1 abnimmt. D.h., der erforderliche Flächenaufwand ist abhängig von der Anzahl der Eingaben, für die die Fehlertoleranz garantiert werden soll. Je weniger Eingaben fehlertolerant verarbeitet werden, desto höher ist die Kostenersparnis im Bezug auf die Fläche des Gesamtsystems. Wenn Fehlertoleranz für 50% der Eingaben garantiert wird, beträgt die durchschnittliche Fläche des Beispiels etwa 60% der notwendigen Fläche eines entsprechenden TMR-Ansatzes.
Es ist ersichtlich, dass der Schutz aller Eingaben zu einem TMR-System führt und der Schutz keiner Eingabe zu einer Einzelschaltung $S = S_1$. Der Haupteffekt der Flächenreduzierung zwischen diesen beiden Grenzen ist der Schaltung s_2 zuzuschreiben. Das gilt besonders dann, wenn in X_1 wenige Werte enthalten sind. Der Grund dafür ist, dass die Optimierung der Schaltung s_3 von der Optimierung der Schaltung s_2 abhängt. Die Möglichkeiten zur Optimierung von s_3 sind somit eingeschränkt, denn die Funktion $s_3(x)$

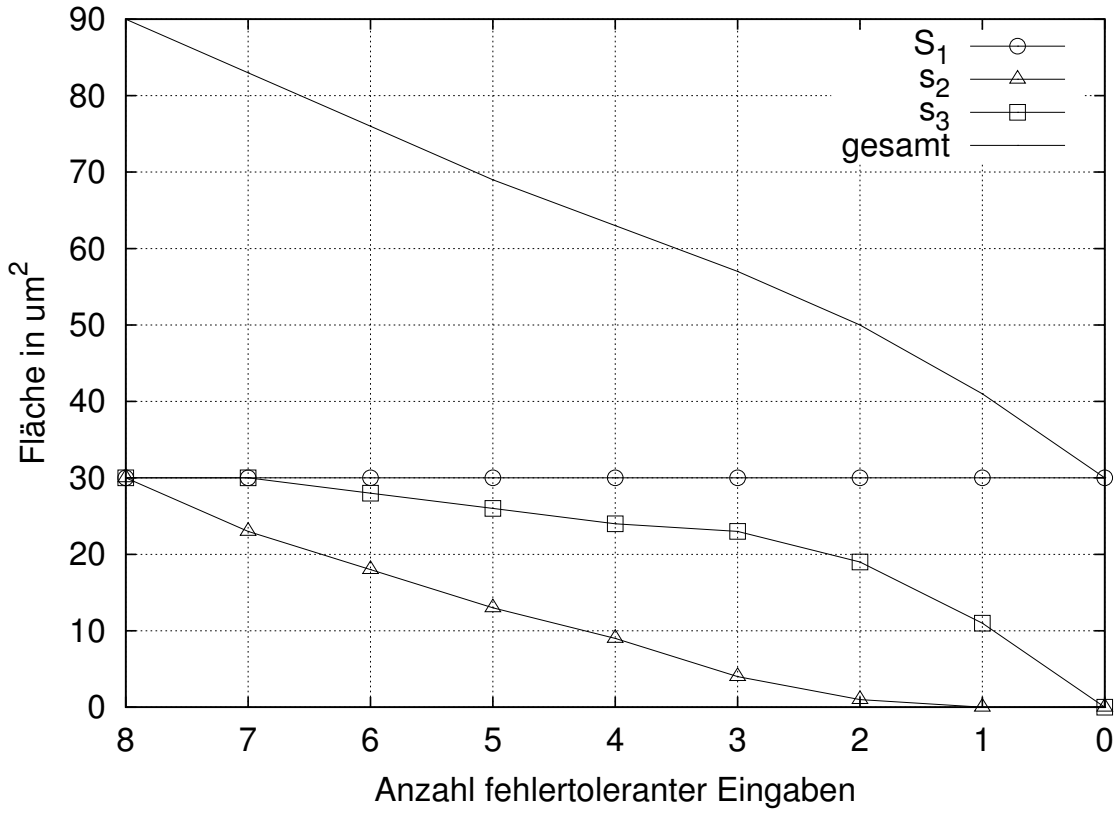

Abbildung 4.2.: Durchschnittlicher Flächenaufwand pro Anzahl geschützter Eingaben

kann nicht mehr don't-care Werte aufweisen als die Funktion $s_2(x)$. Im Allgemeinen ist die Anzahl der don't-care Werte in $s_2(x)$ also größer als in $s_3(x)$, wodurch die Schaltung s_2 besser optimiert werden kann. Ist die Anzahl der don't-care Werte in $s_2(x)$ und $s_3(x)$ gleich, dann entspricht die optimierte Schaltung s_2 der Schaltung S_1, was auch für s_3 gilt. Das Ergebnis ist dann wieder eine TMR-Realisierung.
Da sich der erforderliche Flächenaufwand von s_3 wesentlich geringer auf die Reduzierung der Gesamtfläche auswirkt und s_3 selbst beim Schutz von 25% aller möglichen Eingaben der Größe von S_1 entsprechen kann, ist in Tabelle 4.5 dargestellt, wie oft s_3 in dem kleinen Beispiel die Größe von S_1 annimmt. Allerdings ist in diesen Fällen s_3 nicht immer eine Kopie von S_1. Es wurden lediglich dieselben Gatter verwendet. Bei einer größeren Schaltung S_1 ist daher damit zu rechnen, dass die Optimierung von s_3 meist zu besseren Ergebnissen führen wird.

Tabelle 4.5.: Häufigkeit mit der s_3 die Fläche von S_1 einnimmt

geschützte Eingaben	0	1	2	3	4	5	6	7	8
Häufigkeit in %	0	0	14.29	32.14	51.43	64.29	82.14	100	100

4.1.2. Heuristik für Netzlisten

Kombinatorische Digitalschaltungen werden in der Regel durch eine Auflistung von Ein- und Ausgabevektorpaaren beschreiben. Die maximale Anzahl dieser Vektorpaare beträgt bei einer Schaltung S_1, die die Boolesche Funktion $S_1(x) = y$ realisiert, 2^m. Die Variable m steht dabei für die Eingangswortbreite Schaltung S_1. Für Schaltungen mit großem m ist die Anzahl aller möglichen Ein- und Ausgabevektorpaare somit sehr hoch. Besteht für eine solche Schaltung mit großem m die Menge der zu schützenden Eingaben X_1 aus wenigen Vektoren, dann werden nach dem zuvor beschriebenen Verfahren einer großen Menge an Ausgaben don't-care Werte zugeordnet, die durch den Optimierungsprozess beliebige Werte annehmen. Für alle diese Werte muss allerdings bestimmt werden, ob die Ausgabe der Schaltung S_1 ungleich der Ausgabe der Schaltung s_2 ist. Dieser Prozess ist zwar parallelisierbar, er erfordert aber dennoch einen hohen Rechenaufwand.
Wie es sich schon an dem kleinen Beispiel aus dem vorangegangenen Abschnitt zeigte, ist der Einfluss der Schaltung s_3 auf die Reduzierung der Fläche des Gesamtsystems deutlich geringer als der der Schaltung s_2. Für die Erzeugung der Schaltung s_3 kann daher auch eine Heuristik eingesetzt werden, mit der sich die Funktion $s_3(x)_{opt}$ möglichst gut annähern lässt, solange ihre Realisierung als Schaltung nicht mehr Fläche einnimmt als die Schaltung S_1.
Eine einfache Möglichkeit dieses Problem zu umgehen läge darin, $s_3(x) = S_1(x)$ zu setzen. Dann trägt allerdings nur noch die Minimierung von $s_2(x)$ zur Flächenreduzierung bei. Da aber die Funktionen $S_1(x)$ und $s_2(x)$ in jedem Fall als Netzlisten implementiert werden müssen, können sie auch zur Erzeugung der Schaltung s_3 verwendet werden. Eine Möglichkeit, die Netzlisten geeignet zu verknüpfen, wird nachfolgend beschrieben.
Es sei $\chi(x)$ die charakteristische Funktion von X_1, welche von dem Entwerfer grundsätzlich angegeben werden muss, da er festlegt, welche Eingaben fehlertolerant verarbeitet werden sollen.

$$\chi(x) = \begin{cases} 1 & \text{für} \quad x \in X_1 \\ 0 & \text{sonst} \end{cases}$$

Die entsprechende Schaltungsrealisierung bzw. Netzliste von $\chi(x)$ wird mit χ bezeichnet. Außerdem wird festgelegt, dass die Ausgabe von s_3 gleich der Ausgabe von S_1 sein muss, wenn

1. $\chi(x) = 1$

oder wenn

2. $\chi(x) = 0$ und $s_2(x) \neq S_1(x)$.

Werden nun noch alle nicht spezifizierten Werte von $s_3(x)$ auf $s_3(x) = 0$ gesetzt und wird berücksichtigt, dass $s_2(x) \neq S_1(x)$ ist, genau dann, wenn $S_1(x) \oplus s_2(x) = 1$ ist, dann entspricht diese Verknüpfung der Funktion

$$(((s_2(x) \oplus S_1(x)) \wedge \overline{\chi}(x)) \vee \chi(x)) \wedge S_1(x) = S_1(x) \wedge (\overline{\overline{\chi}(x) \wedge s_2(x)}) = s_3(x),$$

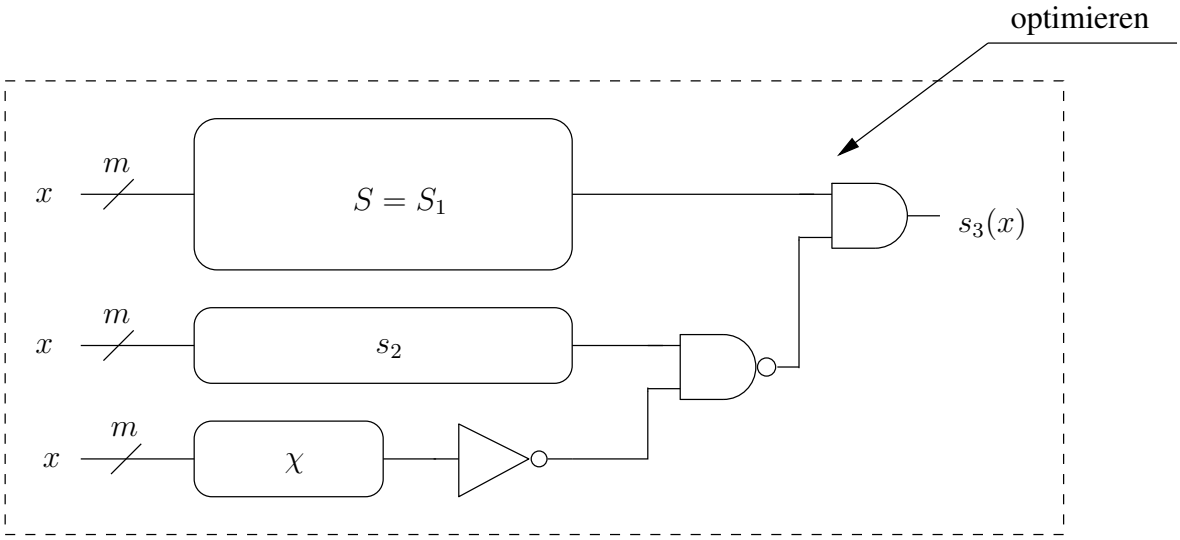

Abbildung 4.3.: Erzeugung von s_3 aus den Netzlisten von S_1, χ und s_2

welche in vereinfachter Form in Abbildung 4.3 dargestellt ist.
Die jeweilige Ausgabe der Funktion $s_3(x)$ kann der zugehörigen Wertetabelle 4.6 entnommen werden. Dabei gibt $\chi(x) = 0$ an, dass die Eingabe der unkritischen Teilmenge $X \setminus X_1$ zuzuordnen ist und $\chi(x) = 1$ zeigt, dass die aktuelle Eingabe Element der kritischen Teilmenge X_1 ist.

Tabelle 4.6.: Wertetabelle der Netzlistenverknüpfung

$S_1(x)$	$\chi(x)$	$s_2(x)$	$s_3(x)$	Bemerkung
0	0	0	0	$x \notin X_1$ und $S_1(x) = s_2(x) \rightarrow$ setze $s_3(x) = 0$
0	0	1	0	$x \notin X_1$ und $S_1(x) \neq s_2(x) \rightarrow$ setze $s_3(x) = S_1(x)$
0	1	0	0	$x \in X_1 \rightarrow$ setze $s_3(x) = S_1(x)$
0	1	1	0	$x \in X_1 \rightarrow$ setze $s_3(x) = S_1(x)$
1	0	0	1	$x \notin X_1$ und $S_1(x) \neq s_2(x) \rightarrow$ setze $s_3(x) = S_1(x)$
1	0	1	0	$x \notin X_1$ und $S_1(x) = s_2(x) \rightarrow$ setze $s_3(x) = 0$
1	1	0	1	$x \in X_1 \rightarrow$ setze $s_3(x) = S_1(x)$
1	1	1	1	$x \in X_1 \rightarrow$ setze $s_3(x) = S_1(x)$

Bei allen unkritischen Eingaben, für die sich die Ausgaben der Funktionen $S_1(x)$ und $s_2(x)$ unterscheiden, wird $s_3(x)$ die Ausgabe der Funktion $S_1(x)$ zugeordnet. Ansonsten wird der Ausgabe von $s_3(x)$ im Fall einer Eingabe aus $X \setminus X_1$ der Wert 0 zugewiesen. Für kritische Eingaben $x \in X_1$ wird $s_3(x)$ auf den Wert von $S_1(x)$ gesetzt, damit alle drei Funktionen dieselben Ausgaben liefern. Die Zeilen 4 und 7 sind aufgrund der Ausgangsbedingungen für den Entwurf der Schaltung s_2 in der Realität nicht relevant.

4.1.3. Experimentelle Ergebnisse

In diesem Abschnitt wird für das vorgeschlagene präzise Verfahren ermittelt, wie sich die Anzahl der Eingaben $x \in X_1$, für die Fehlertoleranz garantiert werden soll, auch bei größeren Schaltungen auf die Fläche der redundanten Systemteile auswirkt.
Für diese Untersuchungen wurden aus der Benchmark-Suite LGSynth91 vier größere kombinatorische Schaltungen im PLA-Format ausgewählt, an denen die vorgestellte Methode einschließlich der Steuerfunktion $\chi(x)$ implementiert wurde. Die Wortbreiten m der Eingabevekoren unterscheiden sich wie folgt: xor5 ($m = 5$), Z9sym ($m = 9$), t481 ($m = 16$), parity ($m = 16$). Zur Vereinfachung wurden Schaltungen mit einem Ausgang ausgewählt. Wie zuvor beschrieben, kann die Methode aber auch auf Schaltungen mit mehreren Ausgängen angewendet werden.
Die Menge X_1 wurde so gewählt, dass sie pro Stufe die jeweils ersten $p\%$ an Eingaben aus der geordneten Menge aller Binärvektoren $X = \{0, 1\}^m$ enthält. Pro Stufe nimmt p um 10% zu. Kombinationen mit anderen Eingaben wurden bei den Benchmark-Schaltungen aufgrund der Komplexität ihrer Eingabemengen nicht vorgenommen. Da die Schaltungen untereinander keine Ähnlichkeiten besitzen, zeigen die Stichproben dennoch, welche Möglichkeiten sich durch die spezifische Fehlertoleranz bezüglich der Flächenreduzierung für größere Schaltungen ergeben.
Bei der Synthese wurden zunächst die im PLA-Format vorliegenden Ausgangsschaltungen mit dem Espresso-Algorithmus optimiert. Anschließend wurden auf dieser Grundlage die PLA-Beschreibungen der zusätzlichen Schaltungen s_2, s_3 und χ erstellt, welche ebenfalls mit dem Espresso-Algorithmus optimiert wurden. Die PLA-Beschreibungen aller Teilschaltungen wurden dann mit dem Synopsys Design Compiler auf die schon zuvor verwendete 0.13 μm Standardzellenbibliothek abgebildet. Aus dieser Bibliothek wurden ausschließlich Inverter ($\sim 5\mu m^2$), UND-Gatter mit zwei Eingängen ($\sim 8\mu m^2$) sowie ODER-Gatter mit zwei Eingängen ($\sim 8\mu m^2$) genutzt, damit für weitere Untersuchungen bessere Vergleichsmöglichkeiten bestehen. Bei der Abbildung erfolgte keine weitere Optimierung durch den Design Compiler, um nur den Effekt der Minimierung der partiell definierten Booleschen Funktionen zu zeigen.
Die Ergebnisse sind in den Tabellen 4.7 bis 4.10 dargestellt. Sie beinhalten die Flächen aller Teilschaltungen bis auf die Voter und zeigen, dass die Methode auch bei größeren Schaltungen effektiv eingesetzt werden kann, um den Flächenaufwand von TMR-Systemen zu reduzieren. Der ausschlaggebende Faktor für die Reduzierung der erforderlichen Fläche des Gesamtsystems ist erwartungsgemäß die Optimierung der Schaltung s_2. Es zeigt sich aber auch, dass für die Flächenreduzierung der Einfluss der Schaltung s_3 steigt. Im Gegensatz zu dem sehr kleinen Beispiel aus Abschnitt 4.1.1 ist die Größe der Schaltung s_3 nun nicht immer gleich der Größe der Schaltung S_1, auch wenn die Anzahl der zu schützenden Eingaben nahe bei 100% liegt. Das gilt bei den untersuchten Benchmark-Schaltungen insbesondere dann, wenn ihre Eingabemenge X sehr groß ist.
In Abbildung 4.4 ist für jedes Gesamtsystem die prozentuale Reduzierung des Flächenaufwandes im Vergleich zu einer entsprechenden TMR-Realisierung dargestellt. Aus den verhältnismäßig ähnlichen Kurvenverläufen der grundlegend verschiedenen Schaltungen ist zu erkennen, dass das Flächeneinsparpotential der vorgestellten Methode

Tabelle 4.7.: Messergebnisse der Benchmark-Schaltung xor5

Mindestanzahl geschützter Eingaben in %	Schaltungsflächen in μm^2				
	S_1	s_2	s_3	χ	gesamt
0	745	0	0	0	745
10	745	22	239	32	1038
20	745	94	506	24	1369
30	745	191	430	32	1398
40	745	269	745	16	1775
50	745	272	745	0	1762
60	745	476	699	32	1952
70	745	584	705	24	2058
80	745	689	745	32	2211
90	745	745	745	0	2235
100	745	745	745	0	2235

Tabelle 4.8.: Messergebnisse der Benchmark-Schaltung Z9sym

Mindestanzahl geschützter Eingaben in %	Schaltungsflächen in μm^2				
	S_1	s_2	s_3	χ	gesamt
0	6025	0	0	0	6025
10	6025	785	2055	490	9335
20	6025	1471	2889	656	11041
30	6025	2340	3620	957	12942
40	6025	2886	4142	710	13763
50	6025	3470	4142	925	14562
60	6025	4155	5223	455	15858
70	6025	4621	5519	487	16652
80	6025	5059	5885	231	17200
90	6025	5352	5885	56	17318
100	6025	6025	6025	0	18075

Tabelle 4.9.: Messergebnisse der Benchmark-Schaltung t481

Mindestanzahl geschützter Eingaben in %	Schaltungsflächen in μm^2				
	S_1	s_2	s_3	χ	gesamt
0	48742	0	0	0	48742
10	48742	382	3475	8	52607
20	48742	12953	18415	724	80834
30	48742	20341	25588	2253	96924
40	48742	24195	30208	1321	104466
50	48742	35117	39661	371	123891
60	48742	35117	39661	371	123891
70	48742	35117	39822	229	123910
80	48742	36462	39822	347	125373
90	48742	42558	43472	659	135431
100	48742	48742	48742	0	146226

Tabelle 4.10.: Messergebnisse der Benchmark-Schaltung parity

Mindestanzahl geschützter Eingaben in %	Schaltungsflächen in μm^2				
	S_1	s_2	s_3	χ	gesamt
0	4398987	0	0	0	4398987
10	4398987	396391	1852151	121	6647650
20	4398987	845569	2944069	984	8189609
30	4398987	1314102	3433158	2695	9148942
40	4398987	1801893	4163210	2173	10366263
50	4398987	2076923	4398987	0	10874897
60	4398987	2788920	3645647	121	10833675
70	4398987	3287546	4088066	121	11774720
80	4398987	3803615	4024386	105	12227093
90	4398987	4337214	4211687	113	12948001
100	4398987	4398987	4398987	0	13196961

relativ gleichbleibend ist. Bei jedem Benchmark verringert sich die redundante Fläche, je weniger Eingaben zu schützen sind. Werden ca. 30% aller Eingaben geschützt, können gegenüber einem TMR-Ansatz durchschnittlich 33% an Fläche eingespart werden. Das entspricht einer kompletten Teilschaltung eines TMR-Systems.

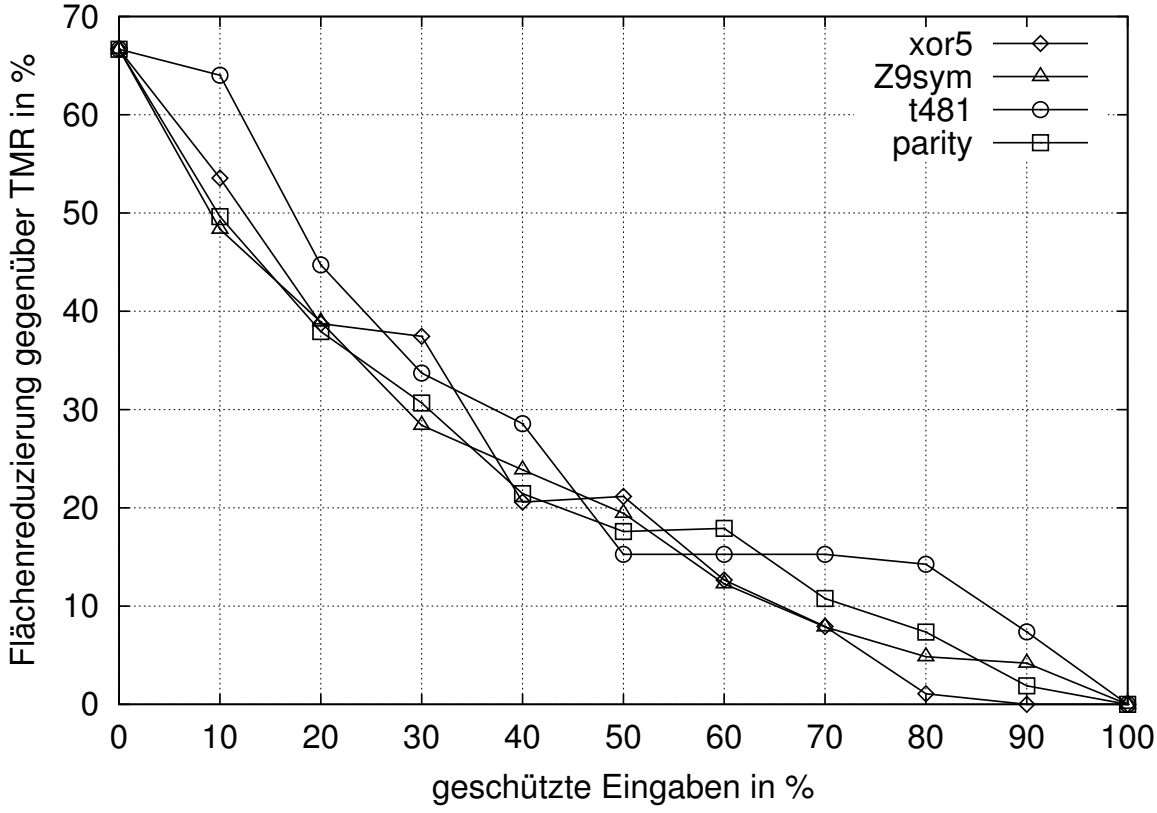

Abbildung 4.4.: Flächenreduzierung im Vergleich zu TMR

Der Mehraufwand für die Steuerfunktion $\chi(x)$ ist in den betrachteten Beispielen gering. Durchschnittlich benötigte die Schaltung χ ca. 1% der Gesamtfläche. Das Maximum lag in nur zwei Fällen bei 5%. Generelle Aussagen über den Aufbau von χ lassen sich durch den Versuchsaufbau allerdings nicht treffen, da die Fläche der Schaltung durchaus davon abhängt, welche Eingabevektoren in X_1 enthalten sind. Besteht X_1 z.B. aus den ersten 50% aller geordneten Binärvektoren der Form $(x_1, \ldots, x_m)$ und ist x_m die höchstwertige Komponente, so gilt immer $\chi(x) = x_m$. Die Steuerfunktion kann dann entsprechend durch eine einzelne Leitung implementiert werden. Das zeigt sich auch in den Ergebnissen der Schaltungen xor5 und parity. Im Fall von Z9sym und t481 konnten bei der 50%-Stufe nach der Optimierung von s_3 noch zusätzliche Vektoren in X_1 aufgenommen werden, wodurch die Steuerfunktion eine größere Fläche benötigte. Für Schaltungen mit mehr als einem Ausgang wird die Größe der Schaltung χ weiter vernachlässigt werden können, da sie lediglich von der Anzahl der Eingabevektoren abhängt. Die Breite der Ausgabevektoren ist für χ unerheblich.

Damit wurde experimentell gezeigt, dass das vorgeschlagene Verfahren, die Fehlertoleranz auf die tatsächlich benötigten Anforderungen abzustimmen, dazu verwendet werden kann, die Fläche von TMR-Systemen erheblich zu reduzieren.

4.2. Erzeugung größtmöglicher On- und Off-Sets

In dem vorangegangenen Abschnitt 4.1 wurde bereits angedeutet, dass es bei Schaltungen mit großer Eingabewortbreite aufgrund der großen Anzahl an möglichen Eingabezuweisungen sehr aufwändig sein kann, die Funktionen der zusätzlichen Schaltungen s_2 und s_3 über das Wertetabellenverfahren zu bestimmen. Zum einen basiert die Bestimmung der Unterschiede zwischen den Ausgaben von S_1 und s_2 auf sehr rechenintensiven Schaltungssimulationen und zum anderen steigt der Aufwand für die Minimierung partiell definierter Boolescher Funktionen mit der Eingabewortbreite.
Es wurde gezeigt, wie die Schaltung s_3 aus den Netzlisten der Schaltungen S_1, s_2 und χ bestimmt werden kann. Es besteht aber für die Schaltung s_2 noch das Problem, dass gängige Entwurfswerkzeuge bei der Synthese don't-care Werte in den Ausgaben einer Schaltung nicht optimal verarbeiten, da es im industriellen Entwurfsprozess unüblich ist, unbestimmte Werte am Ausgang einer Schaltung bereitzustellen. So werden die don't-care Werte z.B. durch vorbestimmte Werte oder schnell zu bestimmende Komponenten ersetzt [RS09], was im Bezug auf den Schaltungsaufwand bei der spezifischen Fehlertoleranz meist nicht von Vorteil ist.
In diesem Abschnitt wird daher eine Heuristik vorgestellt, die es erlaubt, das Verfahren der spezifischen Fehlertoleranz auch ohne die Verwendung von don't-care Werten oder Schaltungssimulationen umzusetzen. Sie kann mit gängigen Hardwarebeschreibungssprachen implementiert werden und entsprechende Schaltungsbeschreibungen lassen sich ohne Einschränkungen auch durch industriell verwendete Werkzeuge im Schaltungsentwurf synthetisieren.
Ausgangspunkt für diese Heuristik ist die charakteristische Funktion χ der kritischen Eingabemenge X_1. Diese Funktion wird im Entwurfsverfahren von Schaltungen mit spezifischer Fehlertoleranz immer vom Entwerfer angegeben, da er bestimmt, welche Eingaben $x \in X$ der Ausgangsschaltung $S = S_1$ fehlertolerant ausgelegt werden sollen. Die Schaltungen χ und $S = S_1$ können in Form von Netzlisten oder als Verhaltensbeschreibungen z.B. in Verilog oder VHDL vorliegen. Eine Verhaltenssimulation der Ausgangsschaltung $S = S_1$ oder anderer Schaltungen ist nicht notwendig.
Die Funktionen der Systeme s_2 und s_3 lassen sich durch folgende Verknüpfungen von χ mit $S_1(x)$ erzeugen:

$$s_2(x) = S_1(x) \wedge \chi(x) \tag{4.3}$$

und

$$s_3(x) = S_1(x) \vee \overline{\chi}(x). \tag{4.4}$$

Für kritische Eingaben $x \in X_1$, d.h. für Eingaben, bei denen $\chi(x) = 1$ ist, gilt damit

$$s_2(x) = s_3(x) = S_1(x)$$

und Gleichung 3.1 ist stets erfüllt. Für alle übrigen Eingaben $x \in X \setminus X_1$ ist die Ausgabe von $\chi(x) = 0$ und es gilt

$$s_2(x) = 0 \quad \text{und} \quad s_3(x) = 1.$$

Auf diese Weise wird sichergestellt, dass sich die Ausgaben der Systeme s_2 und s_3 für alle Eingaben $x \in X \setminus X_1$ unterscheiden. Entsprechend ist mindestens einer der Werte $s_2(x) = 0$ oder $s_3(x) = 1$ gleich der Ausgabe der Funktion $S_1(x)$ und Gleichung 3.2 ist erfüllt.
Im Hinblick auf Schaltungen mit mehr als einem Ausgang ist Gleichung 3.2 für jede Komponente k des Ausgabevektors y erfüllt.
Zur Optimierung werden im Gegensatz zur Variante mit den partiell definierten Booleschen Funktionen keine don't-care Werte verwendet, da die zusätzlichen Schaltungen s_2 und s_3 von vornherein vollständig definiert sind. Der Funktion $s_2(x)$ wird dabei ein möglichst großes Off-Set und der Funktion $s_3(x)$ ein möglichst großes On-Set zugewiesen. Es wird angenommen, dass sich diese Verteilung der Ausgabewerte positiv auf die Reduzierung der Fläche der optimierten Schaltungen s_2 und s_3 auswirkt.
Sind $S_1(x)$ und $\chi(x)$ als Netzlisten bestimmt worden, so können die Schaltungen der Funktionen $s_2(x)$ und $s_3(x)$ durch Optimierung der in Abb. 4.5 dargestellten Schaltungen erzeugt werden.
Eine Realisierungsmöglichkeit in Form einer Verhaltensbeschreibung wird nachfolgend anhand eines Beispiels in der Hardwarebeschreibungssprache Verilog vorgestellt. Das in diesem Zusammenhang verwendete Beispiel basiert auf einer Schaltung S, die in Form einer Verilog-Beschreibung gegeben ist. Die Schaltung hat aus Gründen der Übersichtlichkeit lediglich eine Eingabewortbreite von vier Bit und eine Ausgabewortbreite von einem Bit. In gleicher Weise kann das Verfahren aber auch auf größere Schaltungen übertragen werden.

4.2.1. Implementierungsbeispiel

Die Schaltung $S = S_1$ sei durch das Modul `comp1` als Verilog-Beschreibung gegeben. Eine genauere Definition dieser Komponente wird an dieser Stelle ausgelassen, da das nachfolgende Framework durch Anpassung der Input- und Output-Ports auch auf beliebige andere Schaltungsbeschreibungen angewendet werden kann.
Die zu schützenden Signale $x \in X_1$ der Schaltung $S = S_1$ sind durch die Eingangsbelegungen $(x_1, x_2, x_3, x_4) = \{0000, 00\text{-}1, 1000\}$ bestimmt. Eine Implementierung der Schaltung χ kann dann entsprechend des Quellcodes aus Abbildung 4.6 über das Modul `chi` vorgenommen werden.
In den Zeilen 1-5 wird das Modul `chi` mit seinen zugehörigen Input- und Output-Ports deklariert. Die vier Eingänge der Schaltung werden dem Input-Port `I` zugewiesen und der Ausgang der Schaltung trägt die Bezeichnung `O`. In Zeile 6 wird zusätzlich eine ein

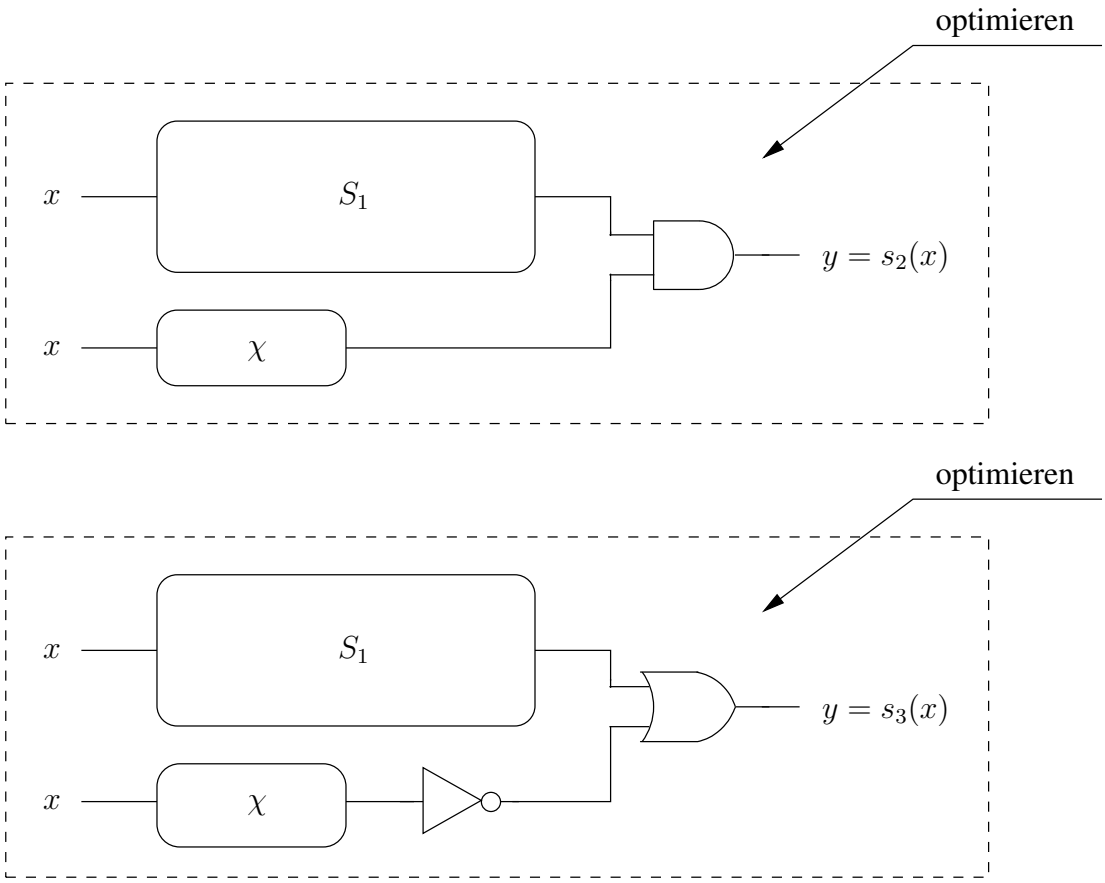

Abbildung 4.5.: Erzeugung von s_2 und s_3 aus den Netzlisten von S_1 und χ

Bit breite Variable `out_tmp` deklariert, welche das Ergebnis der Kombinatorik von `chi` trägt. Diese Variable wird in Zeile 10 dem Ausgang `O` zugewiesen.

Die Hauptaufgabe des Entwerfers besteht darin, den kombinatorischen Prozess von `chi` zu beschreiben. Diese Beschreibung beginnt in Zeile 13. Jede Änderung am Schaltungseingang `I` bewirkt, dass die darauf folgende `casex`-Anweisung in den Zeilen 15-20 ausgeführt wird. Der Entwerfer kann die Menge der kritischen Eingaben aus X_1 in dieser `casex`-Anweisung nacheinander auflisten und ihre zugehörigen Ausgaben auf den logischen Wert `1` festlegen. Für alle anderen Eingaben $x \in X \setminus X_1$ erzeugt `chi` den logischen Wert `0`.

Da die Anzahl an kritischen Eingabezuweisungen sehr groß sein kann, kann der Entwerfer die Menge der kritischen Eingaben auch durch Bereiche angeben. Die effizienteste Lösung dafür ist die Schaltungsbeschreibung über Cubes vorzunehmen, wie es in Zeile 17 gezeigt wird. Diese Beschreibungen können von Synthesewerkzeugen meist besser im Bezug auf den Hardwareaufwand optimiert werden als Schaltungsbeschreibungen, bei denen alle Eingabevektoren einzeln angegeben werden. Das gilt insbesondere dann, wenn eine Schaltung durch sehr viele Vektoren beschrieben wird.

```
1  module chi (I, O);
2
3    // Deklaration von Ports und Variablen
4    input  [3:0] I;
5    output [0:0] O;
6    reg    [0:0] out_tmp;
7
8    // dem Schaltungsausgang das Ergebnis des kombinatorischen
9    // Prozesses zuweisen
10   assign O = out_tmp;
11
12   // Beschreibung des kombinatorischen Prozesses
13   always @ (I)
14
15     casex (I)
16       4'b0000 out_tmp = 1'b1;
17       4'b00x1 out_tmp = 1'b1;
18       4'b1111 out_tmp = 1'b1;
19       default : out_tmp = 1'b0;
20     endcase
21
22 endmodule
```

Abbildung 4.6.: Verilog-Implementierung der Schaltung χ

Die Schaltungen s_2 und s_3 können nun durch die Verknüpfung von Instanzen der Komponenten `chi` und `comp1` beschrieben werden. Ein Beispiel für die Implementierung der Schaltung s_2 ist in Abbildung 4.7 gegeben. Die Beschreibung ist eine direkte Umsetzung der Struktur aus Abbildung 4.5.
Das zugehörige Modul wird mit `comp2`, sein Input-Port mit `I` und sein Output-Port mit `O` bezeichnet (Zeile 1-5). In der weiteren Beschreibung der Zeilen 6-11 wird je eine Instanz der Komponenten `chi` und `comp1` erzeugt, deren Eingänge mit dem Input-Port `I` der Schaltung `comp2` verbunden sind. Die Ausgänge der Instanzen `_chi` und `_comp1` sind entsprechend ihrer Bezeichner mit den Leitungen `chi_out` und `comp1_out` verbunden. Die Leitungen `chi_out` und `comp1_out` tragen somit die Ergebnisse der kombinatorischen Prozesse aus den Schaltungen χ und S_1 und werden in Zeile 12 nach dem Prinzip aus Abbildung 4.5 mit Hilfe eines UND-Gatters verknüpft. Das Ergebnis wird durch die Anweisung aus Zeile 15 dem Schaltungsausgang `O` zugewiesen.
Eine Implementierung des Systems s_3 erfolgt in ähnlicher Weise. Es muss lediglich die Modulbezeichnung angepasst und die Zeile 15 durch die Zuweisung:

```
assign O = (!chi_out || comp1_out);
```

ersetzt werden.
Die Synthese der Teilsysteme S_1, s_2, s_3 und χ muss separat durchgeführt werden. Es ist darauf zu achten, dass in den Komponenten `comp2` und `comp3` sämtliche Hierarchien

```
1  module comp2 ( I, O );
2
3    // Deklaration von Ports und Variablen
4    input  [3:0] I;
5    output [0:0] O;
6    wire chi_out;
7    wire comp1_out;
8
9    // Komponenteninstanziierung
10   chi _chi ( .I(I), .O(chi_out) );
11   comp1 _comp1 ( .I(I), .O(comp1_out) );
12
13   // Verknuepfung der Ausgaben von chi und comp1
14   // Ergebnis wird dem Schaltungsausgang zugewiesen
15   assign O = (chi_out && comp1_out);
16
17 endmodule
```

Abbildung 4.7.: Verilog-Implementierung der Schaltung s_2

entfernt werden, damit die Optimierung ausschließlich auf Basis der kombinatorischen Logik von S_1 und χ erfolgen kann.

4.2.2. Experimentelle Ergebnisse

Die Heuristik zur Erzeugung großer On- und Off-Sets wurde, wie zuvor an Schaltungen aus der Benchmark-Suite LGSynth91 einschließlich der charakteristischen Funktion χ implementiert. Neben den 4 Schaltungen, die auch im Abschnitt 4.1.3 zur Synthese mit dem Espresso-Algorithmus verwendet wurden, wurde das Verfahren an einer weiteren Schaltung implementiert. Die Ergebnisse, die durch das das Synthesewerkzeug Synopsys erzielt werden konnten, sind in den Tabellen 4.11 bis 4.15 dargestellt.
Im Vergleich zur Synthese mit dem Espresso-Algorithmus konnte die Gesamtfläche der einzelnen Schaltungen durch das Synthesewerkzeug Synopsys noch einmal erheblich reduziert werden. Das ist auf die unterschiedlichen Optimierungsstrategien beider Programme zurückzuführen. Während bei der Minimierung partiell definierter Boolescher Funktionen ausschließlich logische Optimierungen vorgenommen wurden, um die resultierenden PLA-Beschreibungen anschließend auf eine Standardzellenbibliothek abzubilden, beinhaltet die Synthese mit dem Synopsys Design Compiler zusätzliche Optimierungen auf Gatterebene. Eine nachträgliche Synthese der Ergebnisse des Espresso-Algorithmuses mit dem Tool Synopsys hätte also bzgl. der Ausgangsschaltungen zu einem gleichen Ergebnis geführt, wodurch die Ausgangsbasis für einen Vergleich mit einer TMR-Lösung in beiden Ansätzen identisch ist.
Die Schaltungen s_2, s_3 und χ wurden in beiden Verfahren auf gleiche Weise erstellt wie die Schaltung S_1. Zudem wurden dieselben Gatter (AND, OR, NOT) der 0.13 μm Bibliothek wie auch zuvor verwendet. Das Verhältnis zwischen den Größen der einzelnen

Tabelle 4.11.: Messergebnisse der Benchmark-Schaltung xor5 für die Heuristik

Mindestanzahl geschützter Eingaben in %	Schaltungsflächen in μm^2				
	S_1	s_2	s_3	χ	gesamt
0	129	0	0	0	129
10	129	64	59	21	273
20	129	69	80	37	315
30	129	129	129	0	387
40	129	115	104	37	385
50	129	118	112	5	364
60	129	129	129	0	387
70	129	129	129	0	387
80	129	129	129	0	387
90	129	129	129	0	387
100	129	129	129	0	387

Tabelle 4.12.: Messergebnisse der Benchmark-Schaltung Z9sym für die Heuristik

Mindestanzahl geschützter Eingaben in %	Schaltungsflächen in μm^2				
	S_1	s_2	s_3	χ	gesamt
0	567	0	0	0	567
10	567	215	199	53	1034
20	567	266	274	69	1176
30	567	381	282	61	1291
40	567	422	325	69	1383
50	567	392	387	5	1351
60	567	424	406	53	1450
70	567	508	371	69	1515
80	567	508	355	61	1491
90	567	508	365	69	1509
100	567	567	567	0	1701

Tabelle 4.13.: Messergebnisse der Benchmark-Schaltung max46 für die Heuristik

Mindestanzahl geschützter Eingaben in %	Schaltungsflächen in μm^2				
	S_1	s_2	s_3	χ	gesamt
0	1328	0	0	0	1328
10	1328	161	153	53	1695
20	1328	301	282	69	1980
30	1328	446	476	61	2311
40	1328	626	621	69	2644
50	1328	742	650	5	2725
60	1328	906	906	53	3193
70	1328	1124	1024	69	3545
80	1328	1108	1073	61	3570
90	1328	1239	1199	69	3835
100	1328	1328	1328	0	3984

Tabelle 4.14.: Messergebnisse der Benchmark-Schaltung t481 für die Heuristik

Mindestanzahl geschützter Eingaben in %	Schaltungsflächen in μm^2				
	S_1	s_2	s_3	χ	gesamt
0	1057	0	0	0	1057
10	1057	247	225	118	1647
20	1057	387	511	110	2065
30	1057	505	820	126	2508
40	1057	642	564	126	2389
50	1057	578	823	5	2463
60	1057	389	917	118	2481
70	1057	381	919	110	2467
80	1057	806	857	126	2846
90	1057	925	933	126	3041
100	1057	1057	1057	0	3171

Tabelle 4.15.: Messergebnisse der Benchmark-Schaltung parity für die Heuristik

Mindestanzahl geschützter Eingaben in %	Schaltungsflächen in μm^2				
	S_1	s_2	s_3	χ	gesamt
0	11694	0	0	0	11694
10	11694	2571	2549	118	16932
20	11694	3084	2880	110	17768
30	11694	4658	4731	126	21209
40	11694	6057	6153	126	24030
50	11694	6188	7154	5	25041
60	11694	8044	7926	118	27782
70	11694	8902	8902	110	29608
80	11694	9919	9900	126	31639
90	11694	11038	10744	126	33602
100	11694	11694	11694	0	35082

Schaltungen ist somit bis auf die Optimierung auf Gatterebene annähernd gewahrt.
Es ist weiter zu erkennen, dass die Schaltung χ kaum Einfluss auf die Größe der fehlertoleranten Gesamtschaltung hat. Nur im Fall der sehr kleinen Schaltung xor5 steigt die Fläche von χ auf bis zu 29% der Ausgangsschaltung S_1 an.
Die Reduzierung des Hardwareaufwandes, die im Vergleich zu einem TMR-Ansatz durch die vorgestellte Heuristik und das Synthesewerkzeug Synopsys erzielt wurde, ist in Abbildung 4.8 dargestellt.
Die Menge X_1 der geschützten Eingabevektoren besteht in dieser Abbildung wieder in jedem Schritt aus den ersten p% der Eingabevektoren $x = \{0,1\}^m \in X$, wobei X die geordnete Menge aller möglichen Eingabevektoren einer jeweiligen Schaltung ist. Pro Schritt wird p um 10% erhöht.
Die Ergebnisse der Schaltungen Z9sym, max46, t481 und parity zeigen, dass das Prinzip der spezifischen Fehlertoleranz durch die vorgeschlagene Heuristik auch im industriellen Entwurfsprozess erhebliche Kosteneinsparungen mit sich bringen kann. Einzig bei der sehr kleinen Schaltung xor5 führte die Heuristik dazu, dass die Schaltungen s_2 und s_3 in den Schritten 60%, 80% und 90% größer wurden als das Ausgangssystem S_1 und somit eine TMR-Implementierung als Alternative gewählt wurde. Es ist daher günstiger, bei sehr kleinen Schaltungen auf das Verfahren mit den partiell definierten Booleschen Funktionen zurückzugreifen. Es stellt für diese Fälle auch keinen besonderen Aufwand dar.
Die genauere Analyse der Ergebnisse der Schaltung xor5 zeigte zusätzlich, dass sich die vorgeschlagene Heuristik im Fall von 30% und 70% prinzipiell dazu eignet, den Aufwand bzgl. s_2 und s_3 zu reduzieren. Erst das Hinzufügen der Schaltung χ führt dazu, dass ein TMR-Ansatz kostengünstiger ist.
Allgemein kann aus den Ergebnissen aller Benchmarks festgestellt werden, dass sich beim Schutz von 20% aller möglichen Eingaben durchschnittlich die Fläche eines ge-

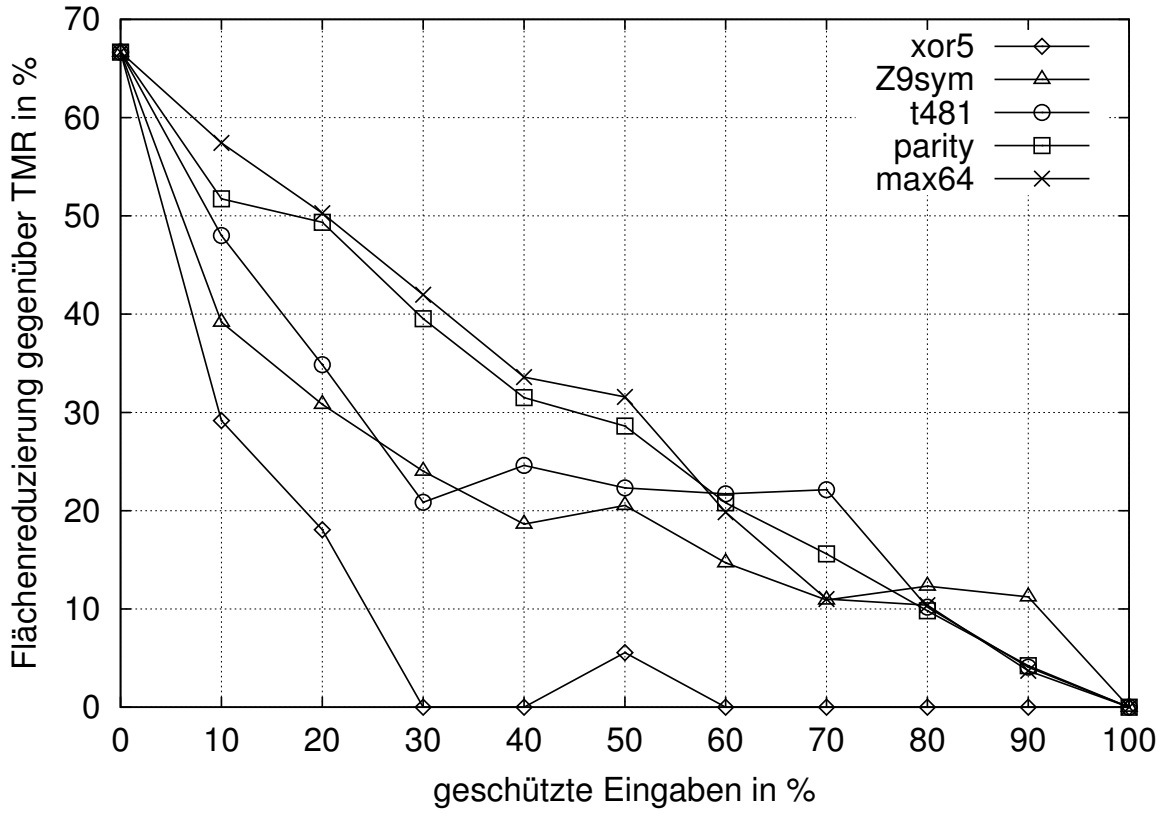

Abbildung 4.8.: Flächenreduzierung gegenüber TMR

samten Teilsystems gegenüber einem TMR-Ansatz einsparen lässt. Damit verschiebt sich dieser Schwellwert im Vergleich zur Variante mit den partiell definierten Booleschen Funktionen um -10%. Durch die vorgeschlagene Heuristik können nun aber auch Schaltungen mit sehr großer Eingabewortbreite nach dem Prinzip der spezifischen Fehlertoleranz entworfen werden, bei denen 20% aller Eingaben noch immer eine sehr große Menge darstellt.
Durch die Heuristik ist die spezifische Fehlertoleranz weiterhin entlang der gesamten Eingabemenge skalierbar und ergibt beim Schutz von etwa 10-50% aller möglichen Eingaben eine Flächeneinsparung von ca. 45% bis 20% gegenüber TMR.
In Abbildung 4.9 sind die durchschnittlichen Flächeneinsparungen der vorgeschlagenen Heuristik im Vergleich zu den Ergebnissen der partiell definierten Booleschen Funktionen dargestellt.
Es wurden für die Erzeugung großer On- und Off-Sets zwei Kurven dargestellt, damit ein Vergleich nicht durch die zusätzlich untersuchte Schaltung max46 beeinflusst wird. Die Kurve "vorgeschlagene Heuristik (4 Schaltungen)" basiert auf denselben Schaltungen, aus denen auch die Ergebnisse der Untersuchungen mit dem Espresso-Algorithmus gewonnen wurden.
Werden durch die vorgeschlagene Heuristik bis zu 40% aller möglichen Eingaben ge-

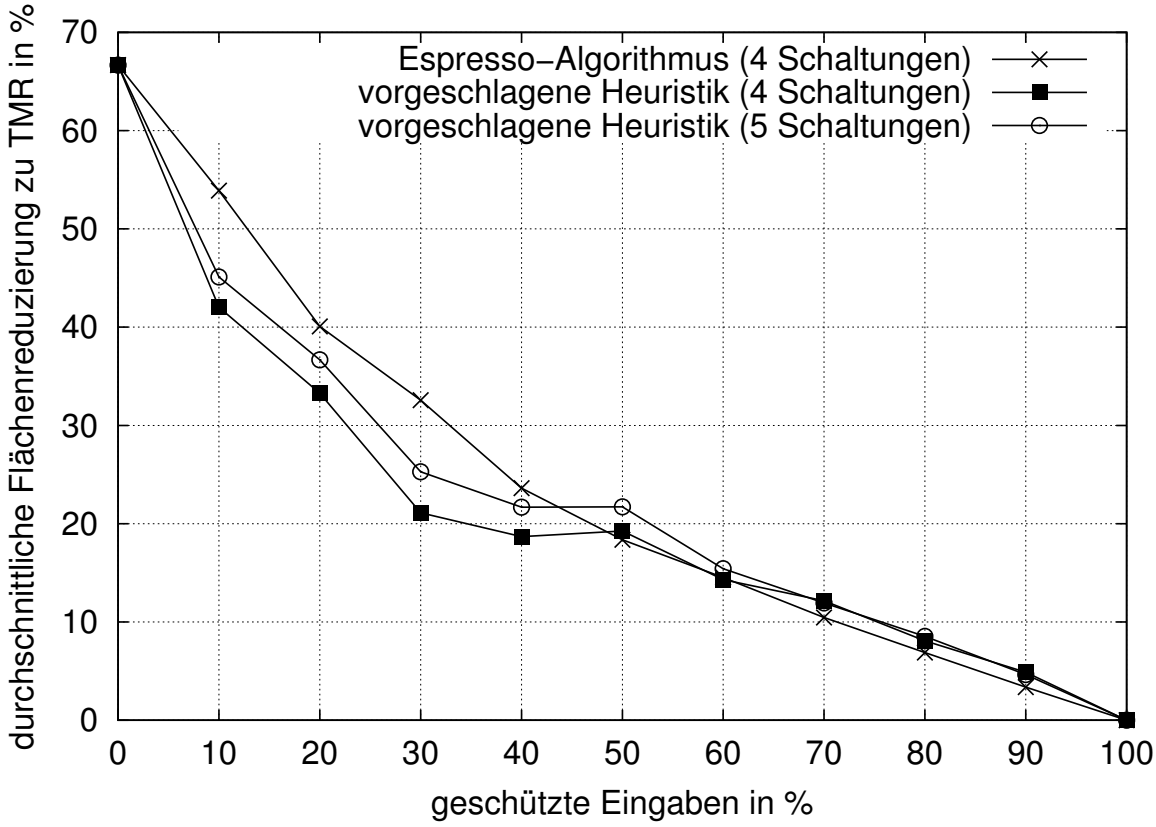

Abbildung 4.9.: Vergleich der durchschnittlichen Flächenreduzierung

schützt, liegt die durchschnittliche Flächeneinsparung nach der Synthese durch das Tool Synopsys etwa 10% unter den Ergebnissen des Espresso-Algorithmuses. Ab dem Schutz von 50% aller Eingaben sind die prozentualen Unterschiede zwischen beiden Varianten nur noch minimal.
Dass sich durch die Heuristik ab dem Schutz von mehr als 50% aller Eingaben teilweise mehr Fläche einsparen lässt als beim Espresso-Algorithmus, ist auf die verschiedenen Optimierungsstrategien der verwendeten Synthesewerkzeuge zurückzuführen.

4.3. Schaltungsbeschreibungen mit Cubes

Während der vorangegangenen Experimente zeigte sich, dass die spezifische Fehlertoleranz bei Schaltungen, die in Form großer Cubes gegeben sind, nur dann durchgehend zu großen Flächeneinsparungen führt, wenn die zusätzlichen Schaltungen s_2 und s_3 ebenfalls durch Cubes beschrieben werden. Das gilt insbesondere dann, wenn sehr viele Eingabevektoren fehlertolerant ausgelegt werden. Die Eingaben der Menge X_1 müssen in diesem Fall möglichst optimalen Cubes zugeordnet werden. Optimal bedeutet in diesem Zusammenhang, dass die Cubes einerseits mindestens alle Eingaben aus X_1 überdecken und

andererseits dazu führen, dass die Schaltungen s_2 und s_3 nach der Synthese weniger Fläche benötigen als die Schaltung S_1.
Das kann dadurch erreicht werden, dass jedem zu schützenden Eingabewert $x \in X_1$ ein Cube aus der Schaltungsbeschreibung von S_1 zugeordnet wird. Entsprechend der in Abschnitt 4.2 vorgestellten Heuristik kann die Schaltung s_2 dann durch genau die Cubes beschrieben werden, mit denen Eingaben aus dem On-Set der Schaltung S_1 geschützt werden sollen. Für alle übrigen Eingaben erzeugt die Schaltung s_2 den logischen Ausgabewert 0. Die Beschreibung der Schaltung s_3 wird anschließend in ähnlicher Weise vorgenommen, wobei sie lediglich jene Cubes enthält, die zu schützende Eingaben aus Off-Set der Schaltung S_1 überdecken. Für alle übrigen Eingabewerte gibt die Schaltung s_3 den logischen Wert 1 aus. Die zusätzlichen Schaltungen s_2 und s_3 werden durch dieses Verfahren nach der Synthese nicht größer als die ursprünglich gegebene Schaltung S_1.
Ist $S_1(x)$ lediglich durch Cubes aus dem On-Set beschrieben worden und gilt somit für alle anderen Eingabewerte implizit, dass sie dem Off-Set zugeordnet werden können, dann lassen sich die Cubes, die benötigt werden, um eine Eingabe aus dem Off-Set zu schützen, durch Negation der On-Set Funktion und anschießendes Anwenden von Kürzungsregeln bestimmen. In analoger Weise kann vorgegangen werden, wenn die Schaltung S_1 ausschließlich durch Cubes aus ihrem Off-Set beschrieben ist.
Eine Einschränkung dieses Verfahrens ist dann gegeben, wenn die Menge aller Cubes, die das On- und das Off-Set der Schaltung S_1 beschreiben zu groß ist, um mit realistischem Aufwand berechnet werden zu können. Es gibt allerdings auch Spezialfälle für dieses Problem. Ist es z.B. nicht möglich alle Off-Set Cubes aus den On-Set Cubes einer Schaltung S_1 zu berechnen, obwohl jeder einzelne Off-Set Cube ohne großen Rechenaufwand aus der Menge der On-Set Cubes bestimmt werden kann, dann kann zumindest auf eine Teilmenge aller Eingaben aus X das Verfahren der spezifischen Fehlertoleranz mit der zuletzt vorgestellten Heuristik angewendet werden. Die Teilmenge der zu schützenden Eingaben $x \in X_1$ darf dabei allerdings nicht sämtliche Cubes aus dem On- und dem Off-Set der Schaltung S_1 überdecken.
Ein Beispiel für diese Problematik ist durch die Schaltung o64 aus der Benchmark-Suite LGSynth91 gegeben. Sie hat eine Ausgabewortbreite von einem Bit und ist nur über ihr On-Set anhand von 65 Cubes mit einer Eingabewortbreite von 130 Bits beschrieben. Eine Bestimmung aller Off-Set Cubes ist für diese Schaltung nicht möglich. Die Gesamtzahl der Off-Set Cubes beträgt 2^{65}. Diese Aussage kann experimentell geprüft werden, indem das komplexe Ausgangsproblem auf viele kleine Probleme abgebildet wird. Es wird sich im Folgenden auch zeigen, dass prinzipiell jeder Cube des Off-Sets der Schaltung o64 implizit bestimmt werden kann.
Auffällig ist bei der Beschreibung der Schaltung o64 der relativ reguläre Aufbau der Cubes. Sie bestehen aus den Vektoren

$$(x^1, x^{130})_1, (x^n, x^m)_2, (x^{n-1}, x^{m-1})_3 \ldots (x^{n-63}, x^{m-63})_j \quad \text{mit} \quad j, n = 65 \quad \text{und} \quad m = 129.$$

Wird diese Schaltungsbeschreibung systematisch auf Schaltungen mit weniger Eingabecubes abgebildet, lässt sich auch eine auffällige Systematik bei der Bildung der Off-Set Cubes erkennen.

In einem Experiment wurde die Schaltung o64=S_1 nacheinander aus den $j = 65$ gegebenen Cubes aufgebaut und anschließend, insofern es möglich war, die jeweils zugehörigen Off-Set Cubes bestimmt. Im ersten Durchgang bestand das On-Set der Schaltung S_1 somit aus dem Cube 1, im zweiten Durchgang aus den Cubes 1 und 2, im dritten Durchgang aus den Cubes 1, 2 und 3 usw.
Im ersten Durchgang, d.h. für den einzelnen On-Set Cube $(x^1, x^{130})_1$, ergaben sich die zwei Off-Set Cubes $\overline{x}^1$ und $\overline{x}^{130}$. Die Ergebnisse von weiteren drei Versuchen sind in Tabelle 4.16 dargestellt. Die Intervallschreibweise wurde in diesem Zusammenhang gewählt, um die Systematik bei der Bildung der Off-Set Cubes besser zu verdeutlichen.
Es zeigt sich, dass sich die Anzahl der Off-Set Cubes mit jedem Versuch verdoppelt. Wenn n die Anzahl der Cubes im On-Set ist, dann wird das Off-Set somit durch 2^n Cubes beschrieben.
Jeder On-Set Cube besteht aus zwei Literalen und jeder Off-Set Cube wird durch genau so viele Literale beschrieben, wie es Off-Set Cubes gibt. Aus den Versuchen geht weiter hervor, dass bei den Off-Set Cubes alle Literale in negierter Form vorkommen. Für eine implizite Bestimmung eines beliebigen Off-Set Cubes ist es nun noch wichtig herauszufinden, nach welchem Muster diese Cubes gebildet werden. Auch für dieses Problem lässt sich anhand der experimentellen Ergebnisse eine Systematik erkennen.
Jedes der n Literale eines Off-Set Cubes ist die negierte Variante eines Literals eines On-Set Cubes. Bei genauerer Betrachtung stellt sich heraus, dass jeder Off-Set Cube immer nur ein negiertes Literal eines On-Set Cubes enthält und es werden für die n belegbaren Stellen alle möglichen Kombinationen nach diesem Muster gebildet.
Diese Systematik kann weiter verallgemeinert werden. Da beim ersten Versuch nur eine Stelle mit jeweils einem negierten Literal aus dem gegebenen On-Set Cube belegt werden kann, ergeben sich die bereits bekannten Off-Set Cubes $\overline{x}^1$ und $\overline{x}^{130}$. Für den zweiten Versuch können zwei Stellen durch jeweils eines der zwei negierten Literal pro On-Set Cube belegt werden, im dritten Versuch sind es drei Stellen, die durch jeweils eines der zwei negierten Literale pro On-Set Cube belegt werden usw.
Werden die On-Set Cubes in Tabelle 4.16 entsprechend der Spalte mit der Bezeichnung Nr. alphabetisch gekennzeichnet und jeweils das 1. Literal eines On-Set Cubes mit 0 und das 2. Literal mit 1 bezeichnet, dann ergeben sich für die Off-Set Cubes dieselben Anordnungen, wie bei der Auflistung aller möglichen Binärvektoren der Form $\{0, 1\}^n$. Für den zweiten Versuch bestehen die vier Off-Set Cubes somit aus den negierten Literalen (A_0, B_0), (A_0, B_1), (A_1, B_0) und (A_1, B_1). In gleicher Weise können die Off-Set Cubes der weiteren Versuche bestimmt werden.
Es kann somit problemlos ohne Berechnung vorhergesagt werden, dass im 5. Versuch der Cube

$$(A_0, B_0, C_0, D_0, E_0) = (\overline{x}^1, \overline{x}^{62}, \overline{x}^{63}, \overline{x}^{64}, \overline{x}^{65})$$

Bestandteil der Off-Set Cubes sein wird.
Um zu prüfen, ob diese Systematik auch bei weiteren Versuchen mit einer größeren Anzahl an On-Set Cubes Bestand hat, wurde das Problem auf äquivalent aufgebaute Schaltungen mit kleinerer Wortbreite abgebildet. Eine Veränderung gegenüber den

Tabelle 4.16.: Ergebnisse der Annäherung an die Off-Set Cubes der Schaltung o64

Versuch	Nr.	x_1	...	x_{63}	x_{64}	x_{65}	...	x_{127}	x_{128}	x_{129}	x_{130}	y
2	A	1	...	-	-	-	...	-	-	-	1	1
	B	-	...	-	-	1	...	-	-	1	-	1
		0	...	-	-	0	...	-	-	-	-	0
		0	...	-	-	-	...	-	-	0	-	0
		-	...	-	-	0	...	-	-	-	0	0
		-	...	-	-	-	...	-	-	0	0	0
3	A	1	...	-	-	-	...	-	-	-	1	1
	B	-	...	-	-	1	...	-	-	1	-	1
	C	-	...	-	1	-	...	-	1	-	-	1
		0	...	-	0	0	...	-	-	-	-	0
		0	...	-	-	0	...	-	0	-	-	0
		0	...	-	0	-	...	-	-	0	-	0
		0	...	-	-	-	...	-	0	0	-	0
		-	...	-	0	0	...	-	-	-	0	0
		-	...	-	-	0	...	-	0	-	0	0
		-	...	-	0	-	...	-	-	0	0	0
		-	...	-	-	-	...	-	0	0	0	0
4	A	1	...	-	-	-	...	-	-	-	1	1
	B	-	...	-	-	1	...	-	-	1	-	1
	C	-	...	-	1	-	...	-	1	-	-	1
	D	-	...	1	-	-	...	1	-	-	-	1
		0	...	0	0	0	...	-	-	-	-	0
		0	...	-	0	0	...	0	-	-	-	0
		0	...	0	-	0	...	-	0	-	-	0
		0	...	-	-	0	...	0	0	-	-	0
		0	...	0	0	-	...	-	-	0	-	0
		0	...	-	0	-	...	0	-	0	-	0
		0	...	0	-	-	...	-	0	0	-	0
		0	...	-	-	-	...	0	0	0	-	0
		-	...	0	0	0	...	-	-	-	0	0
		-	...	-	0	0	...	0	-	-	0	0
		-	...	0	-	0	...	-	0	-	0	0
		-	...	-	-	0	...	0	0	-	0	0
		-	...	0	0	-	...	-	-	0	0	0
		-	...	-	0	-	...	0	-	0	0	0
		-	...	0	-	-	...	-	0	0	0	0
		-	...	-	-	-	...	0	0	0	0	0

bisherigen Erkenntnissen ergab sich dadurch nicht.
Es konnte somit gezeigt werden, dass mit der spezifischen Fehlertoleranz selbst Schaltungen mit extrem großer Wortbreite für eine beliebig wählbare Teilmenge an Ein- und Ausgabezuweisungen fehlertolerant ausgelegt werden können, selbst wenn die Bestimmung aller On- und Off-Set Cubes nicht explizit möglich ist.

4.4. Schaltungen mit mehreren Ausgängen

In den vorangegangenen Abschnitten dieses Kapitels wurden Entwurfsalgorithmen für die spezifische Fehlertoleranz an Schaltungen mit einem 1-dimensionalen Ausgabevektor erklärt. Hier soll nun der allgemeinere Fall für Schaltungen mit mehr als einem Ausgang betrachtet werden. Es wird dazu im Folgenden die Heuristik aus Abschnitt 4.2 angepasst. Sie ist für Schaltungen mit einem Ausgang effizient umsetzbar und berücksichtigt bereits Besonderheiten der Entwurfswerkzeuge.
Für die 1-dimensionalen Ausgaben der Schaltung s_2 und s_3 gilt nach der Heuristik jeweils

$$s_2(x) = \begin{cases} S_1(x) & \text{wenn } x \in X_1 \\ 0 & \text{sonst} \end{cases} \tag{4.5}$$

bzw.

$$s_3(x) = \begin{cases} S_1(x) & \text{wenn } x \in X_1 \\ 1 & \text{sonst.} \end{cases} \tag{4.6}$$

Um dieses Prinzip auf Boolesche Funktionen mit mehreren Ausgängen der Form $y = (y^1, \ldots, y^n) \in \{0,1\}^n$ zu erweitern, wird die Funktion $S_1(x) = y$ als Menge von n unterschiedlichen Funktionen $S_1^1(x) = y_1^1, \ldots, S_1^n(x) = y_1^n$ mit je einem binären Ausgang betrachtet. Die Formeln 4.5 und 4.6 können dann separat auf die Ausgänge der Schaltungen s_2 und s_3 angewendet werden.
Das heißt, wenn Formel 4.5 auf eine bestimmte Ausgangskomponente von s_2 angewendet wird, muss Formel 4.6 auf dieselbe Ausgangskomponente der Schaltung s_3 angewendet werden und wenn Formel 4.6 auf eine bestimmte Ausgangskomponente der Schaltung s_2 angewendet wird, muss entsprechend Formel 4.5 auf dieselbe Ausgangskomponente der Schaltung s_3 angewendet werden. Diese Heuristik wird als Heuristik A bezeichnet.
Eine andere Möglichkeit, die hier auch als Heuristik B bezeichnet wird, besteht darin, die Formel 4.5 auf alle Ausgangskomponenten $y_2^1, \ldots, y_2^n$ des Ausgabevektors der Schaltung s_2 anzuwenden und Formel 4.6 auf alle Ausgangskomponenten $y_3^1, \ldots y_3^n$ des Ausgabevektors der Schaltung s_3 anzuwenden. Somit ergeben sich für Funktionen der Schaltungen s_2 und s_3 die folgenden Definitionen

$$s_2(x) = \begin{cases} S_1(x) & \text{wenn} \quad x \in X_1 \\ \underbrace{(0, \cdots, 0)}_{n} & \text{sonst,} \end{cases} \tag{4.7}$$

und

$$s_3(x) = \begin{cases} S_1(x) & \text{wenn} \quad x \in X_1 \\ \underbrace{(1, \cdots, 1)}_{n} & \text{sonst.} \end{cases} \tag{4.8}$$

Ein einfaches Beispiel für beide Heuristiken ist in Tabelle 4.17 dargestellt. Darin ist die Schaltung S_1 als Boolesche Funktion mit einer Eingabewortbreite von zwei Bits und einer Ausgabewortbreite von vier Bits gegeben und die Menge der zu schützenden Eingaben ist $X_1 = \{00\}$. Eine Umsetzung von S_1 benötigt nur ein XOR-Gatter, ein NOR-Gatter und einen Inverter.
Im Hinblick auf Heuristik A werden offensichtlich keine Gatter benötigt, um die Schaltung s_2 zu realisieren und Schaltung s_3 kann durch lediglich ein NOR-Gatter und einen Inverter implementiert werden. Mit Heuristik B kann die Schaltung s_2 durch lediglich ein NOR-Gatter und die Schaltung s_3 durch nur ein OR-Gatter realisiert werden. Beide Heuristiken können also den zusätzlichen Flächenaufwand im Vergleich zu einer TMR-Implementierung der Schaltung S_1 reduzieren. Für den Entwerfer ist die Heuristik B dabei am praktischsten.

Tabelle 4.17.: Heuristiken für Funktionen mit mehreren Ausgängen

			Heuristik A		Heuristik B	
x	$S_1(x)$	X_1	$s_2(x)$	$s_3(x)$	$s_2(x)$	$s_3(x)$
00	1100	●	1100	1100	1100	1100
01	0101		1100	0011	0000	1111
10	0010		1100	0011	0000	1111
11	1001		1100	0011	0000	1111

5. Spezifische Fehlertoleranz für sequentielle Schaltungen

In diesem Kapitel wird das Konzept der spezifischen Fehlertoleranz für kombinatorische Schaltungen auf sequentielle Schaltungen abgebildet. Mit Hilfe dieses Verfahrens kann eine sequentielle Schaltung so entworfen werden, dass sie eine beliebig wählbare Teilmenge aller möglichen Eingabesequenzen fehlertolerant verarbeitet. Die Zustände, von denen aus diese Eingabesequenzen geschützt werden, können vom Entwerfer beliebig festgelegt werden.
Sobald eine zu schützende Eingabesequenz in einem vorab bestimmten Zustand angewendet wird, garantiert die Schaltung den gleichen Grad an Fehlertoleranz wie die Systemverdreifachung. Für alle anderen Fälle wird keine Fehlertoleranz garantiert.
Es wird zunächst ein einfaches Verfahren für den Entwurf von sequentiellen Schaltungen mit spezifischer Fehlertoleranz vorgestellt. Im Anschluss wird gezeigt, wie dieses Prinzip praktisch in Verbindung mit der Double Modular Redundancy verbessert werden kann. Durch experimentelle Ergebnisse wird für jedes Verfahren gezeigt, welche Flächenreduzierung sich durch das jeweilige Verfahren gegenüber der herkömmlichen Systemverdreifachung ergibt.

5.1. Vereinfachung der Ausgabefunktion

In diesem Abschnitt wird vorgestellt, wie die kombinatorische Logik der Ausgabefunktion von fehlertoleranten sequentiellen Schaltungen verringert werden kann, indem nur wirklich relevante Zustandsübergänge nach dem Prinzip der Systemverdreifachung fehlertolerant ausgelegt werden.
Dazu wird angenommen, dass eine ursprüngliche, nicht fehlertolerante sequentielle Schaltung mit einem m-dimensionalen binären Eingang und einem n-dimensionalen binären Ausgang durch den endlichen Automaten A modelliert werden kann.
Der Automat A wird üblicherweise durch das Tupel $A = (X, Y, Z, z_0, f, g)$ beschrieben [KJ10]. Dabei bezeichnet

- X das Eingabealphabet,
- Y das Ausgabealphabet,
- Z die Menge der Zustände,
- $z_0 \in Z$ den Initialzustand,

- $f : Z \times X \to Z$ die Zustandsüberführungsfunktion und
- $g : Z \times X \to Y$ die Ausgabefunktion des Automaten.

Für die betrachtete Schaltung besteht das Eingabealphabet $X = \{0,1\}^m$ aus der Menge aller m-dimensionalen Binärvektoren und das Ausgabealphabet $Y = \{0,1\}^n$ aus der Menge aller n-dimensionalen Binärvektoren.
Im Allgemeinen bildet ein Automat A eine Eingabesequenz $p = [x_1, x_2, \ldots, x_l] \in X^*$, bestehend aus Eingabewörtern über dem Alphabet X auf eine Ausgabesequenz $q = [y_1, y_2, \ldots, y_l] \in Y^*$, bestehend aus Wörtern über dem Alphabet Y, ab. Dabei sind X^* und Y^* die Mengen aller (endlichen) Wörter der Alphabete X beziehungsweise Y.
Diese Abbildung der Eingabesequenzen auf Ausgabesequenzen wird durch die erweiterte Ausgabefunktion $g^* : Z \times X^* \to Y^*$ beschrieben. Die erweiterte Ausgabefunktion g^* kann für $p = [x_1, x_2, \ldots, x_l]$ wie folgt definiert werden. Wenn die Eingabesequenz $p = [x_1, x_2, \ldots, x_l]$ auf den Automaten A im Initialzustand $z_0 \in Z$ angewendet wird, wird der Automat A die Ausgabesequenz

$$q = [y_1, y_2, \ldots, y_l] = g^*(z, [x_1, x_2, \ldots, x_l]) \quad \text{mit}$$

$$g^*(z, [x_1, x_2, \ldots, x_l]) = [g(z, x_1),\ g((f(z, x_1), x_2),\ \ldots,\ g(f(\ldots f(f(z, x_1), x_2), \ldots), x_l)]$$

ausgeben.
Durch die Abbildung der spezifischen Fehlertoleranz auf sequentielle Schaltungen soll der Automat A jede der unterschiedlich langen Eingabesequenzen $p^1 = [x_1^1, \ldots, x_{l_w}^1], \ldots, p^h = [x_1^h, \ldots, x_{l_w}^h]$ aus h zu schützenden Eingabesequenzen fehlertolerant verarbeiten, sobald eine dieser Eingabesequenzen auf ihn in einem bestimmten ausgezeichneten Zustand $z_a \in Z$ angewendet wird. Die zu schützende Eingabesequenz wird dann so lange fehlertolerant verarbeitet, bis sie verlassen wird oder beendet ist. Die zu schützenden Eingabesequenzen bilden die Menge P aller möglichen zu schützenden Eingabesequenzen

$$P = \{p^1, \ldots, p^h\},$$

wobei P auch alle Teilsequenzen jeder zu schützenden Eingabesequenz enthält.
Im Hinblick auf eine praktische Anwendung kann das Konzept z.B. dazu verwendet werden, die Fehlertoleranz auf kritische Aufgaben eines Systems zu beschränken. Auf diese Weise könnte z.B. der Schutz von Ruhezuständen in einem Mikrocontroller vermieden werden.
Auch verschiedene ausgezeichnete Zustände mit ihren zugehörigen Mengen geschützter Eingabesequenzen können berücksichtigt werden. Um allerdings die folgenden Notationen so einfach wie möglich zu gestalten, wird im Folgenden von nur einem einzelnen ausgezeichneten Zustand z_a und einer einzelnen zu schützenden Eingabesequenz $p = [x_1, \ldots, x_l]$ ausgegangen.
Zur Realisierung des vorgestellten Prinzips, wird mit Hilfe von zwei zusätzlichen Automaten die Zustandsüberführungsfunktion f verdreifacht und die Ausgabefunktion g

teilweise fehlertolerant implementiert. Anschließend werden die Automaten minimiert. Mit der Verdreifachung der Zustandsüberführungsfunktion f wird sichergestellt, dass das Gesamtsystem den ausgezeichneten Zustand z_a und alle Folgezustände $z_f \in Z$, die bei Eingabe einer kritischen Sequenz aus P eingenommen werden können, sicher erreicht werden. D.h., wenn in der Zustandsüberführungsfunktion eines der Automaten A_1, a_2 oder a_3 zu irgendeinem Zeitpunkt ein Fehler auftritt, werden sich mindestens zwei Automaten im korrekten Zustand befinden und eine zu schützende Eingabesequenz $p \in P$ kann in den Zuständen z_a bzw. z_f fehlertolerant verarbeitet werden. Die Ausgabefunktion g wird dazu nur an den Zustandsübergängen fehlertolerant implementiert, die durch die kritischen Eingabesequenzen aus P abgelaufen werden. Auf diese Weise lassen sich Kosten im Bezug auf die Logik der Ausgabefunktion g einsparen.
In Abbildung 5.1 ist der allgemeine Systemaufbau für Automaten mit spezifischer Fehlertoleranz dargestellt. Darin werden dem ursprünglichen Automaten $A = A_1$, ähnlich der spezifischen Fehlertoleranz für kombinatorische Schaltungen, zwei weitere Automaten a_2 und a_3 hinzugefügt, deren Ausgänge in einen Voter V geleitet werden. Alle drei Automaten erhalten gleichzeitig dieselben Eingaben. Die Ausgaben der Automaten A_1, a_2 und a_3 werden durch den Voter V verarbeitet.

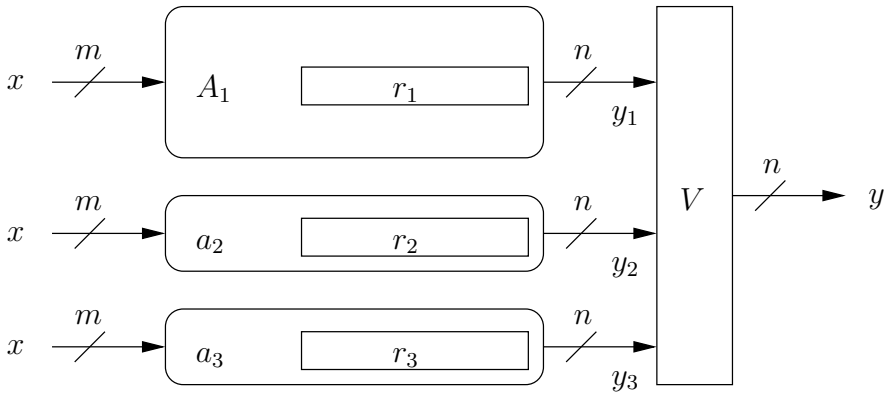

Abbildung 5.1.: Systemaufbau der spezifischen Fehlertoleranz für endliche Automaten

Da die Mengen der Eingaben, Ausgaben und Zustände sowie die Zustandsüberführungsfunktion und die Initialzustände in allen Automaten gleich sind, werden die Automaten a_2 und a_3 vor der Minimierung durch die Tupel $a_2 = (X, Y, Z, z_0, f, g_2)$ und $a_3 = (X, Y, Z, z_0, f, g_3)$ beschrieben. Es unterscheiden sich also nur die Ausgabefunktionen $g = g_1$, g_2 und g_3 voneinander. Die Zustandsüberführungsfunktionen aller drei Automaten sind gleich ($f = f_1 = f_2 = f_3$). Somit ist auch die Anzahl der Zustandsspeicher, die in Abbildung 5.1 durch die Register r_1, r_2 und r_3 symbolisiert werden, in allen Automaten identisch. Der Rest der Automaten sei jeweils kombinatorische Logik zur Berechnung des Folgezustandes und zur Berechnung der Ausgabefunktion.
Als Nächstes wird die Bestimmung der Ausgabefunktionen g_2 und g_3 der Automaten a_2 und a_3 beschrieben. Um die gewünschte Fehlertoleranz für eine zu schützende Eingabesequenz $p \in P$ ab einem Zustand z_a und für alle Folgezustände z_f, die durch die

Eingabesequenz $p = (x_1 \dots x_l)$ erreicht werden, zu erhalten, muss im Bezug auf die Ausgabefunktionen die folgende Bedingung erfüllt sein.

$$g_1^*(z_a, [x_1, \dots, x_l]) = g_2^*(z_a, [x_1, \dots, x_l]) = g_3^*(z_a, [x_1, \dots, x_l]) \tag{5.1}$$

Durch diese Gleichung wird ähnlich der Formel 3.1 sichergestellt, dass das Verhalten der Automaten $a_2 = (X, Y, Z, z_0, f_2 = f, g_2)$ und $a_3 = (X, Y, Z, z_0, f_3 = f, g_3)$ gleich dem Verhalten des ursprünglichen Automaten $A = A_1$ ist, wenn die zu schützende Eingabesequenz p im ausgezeichneten Zustand z_a angewendet wird.
Für alle übrigen Eingaben ist es hinreichend, wenn nur einer der Automaten a_2 oder a_3 komponentenweise dieselbe Ausgabe liefert wie der Automat A_1, da die Ausgaben von einem Mehrheitsentscheider verarbeitet werden. D.h., wenn die i-te Komponente der Ausgabe $y = (y^1, \dots, y^n)$ im Fall einer unkritischen Eingabesequenz durch $g(z, x)_i$ erzeugt wird, dann muss für die Ausgabefunktionen der drei Automaten mindestens die Bedingung

$$g_1(z, x)_i = g_2(z, x)_i \quad \text{oder} \quad g_1(z, x)_i = g_3(z, x)_i \tag{5.2}$$

gelten. Diese Definition ähnelt der Definition aus Formel 3.2. Sie stellt sicher, dass in jedem (erreichbaren) Zustand $z \in Z$ und für jede Eingabe $x \in X$ auf jeder Komponente des Ausgabevektors mindestens einer der zusätzlichen Automaten dieselbe Ausgabe liefert wie der ursprüngliche Automat A_1. Durch diese Bedingung wird keine Fehlertoleranz garantiert.
Da der Zustand z in diesem Fall aber auch ein Zustand z_a oder z_f sein kann, wird jede Eingabe x in einem solchen Zustand immer dann fehlertolerant verarbeitet, wenn sie Teil einer zu schützenden Eingabesequenz p ist.

5.1.1. Entwurfsverfahren

Um die zuvor beschriebenen Definitionen durch ein Verfahren auf eine gegebene sequentielle Schaltung anzuwenden, werden zuerst, ausgehend vom ausgezeichneten Zustand z_a, die Zustände z_1, $z_2, \dots, z_{l-1}$ von A_1 sowie a_2 und a_3 bestimmt, die während der Anwendung der ersten $l-1$ Eingaben aus der zu schützenden Eingabesequenz $p = [x_1, x_2, \dots, x_{l-1}, x_l]$ erreicht werden. Die entsprechenden Zustände sind

$$\begin{aligned} f(z_a, x_1) &= z_1, \\ f(z_1, x_2) &= z_2, \\ &\vdots \\ f(z_{l-2}, x_{l-1}) &= z_{l-1}. \end{aligned}$$

Für diese Zustände und auch für den ausgezeichneten Zustand z_a werden dann auf Basis der Eingaben der zu schützenden Eingabesequenz die identischen Ausgaben der zugehörigen Ausgabesequenzen der einzelnen Automaten definiert. Für die Ausgabefunktionen der Automaten A_1, a_2 und a_3 ergibt sich somit

$$\begin{array}{ccccc} g_1(z_a, x_1) & = & g_2(z_a, x_1) & = & g_3(z_a, x_1), \\ g_1(z_1, x_2) & = & g_2(z_1, x_2) & = & g_3(z_1, x_2), \\ & & \vdots & & \\ g_1(z_{l-1}, x_l) & = & g_2(z_{l-1}, x_l) & = & g_3(z_{l-1}, x_l) \end{array}$$

und die Bedingung aus Formel 5.1 ist für die zu schützende Eingabesequenz p erfüllt.
Damit im Fall einer anderen, noch nicht definierten Eingabe nach Formel 5.2 mindestens einer der zusätzlichen Automaten a_2 oder a_3 auf jeder Komponente des Ausgabevektors dieselbe Ausgabe liefert wie der Automat A_1, kann beispielsweise eine der Heuristiken aus Abschnitt 4.4 auf die Funktionen $g_2(z, x)$ und $g_3(z, x)$ angewendet werden. Nachfolgend wird in diesem Zusammenhang Gebrauch von der Heuristik B gemacht. Für die noch nicht definierten Werte der Funktion $g_2(z, x)$ wird dementsprechend nach Formel 4.7 jede Komponente des n-dimensionalen Ausgabevektors auf den logischen Wert 0 gesetzt und es ergibt sich die Formel

$$g_2(z, x) = \underbrace{0, \ldots, 0}_{n}. \tag{5.3}$$

In ähnlicher Weise werden die noch nicht definierten Werte der Ausgabefunktion $g_3(z, x)$ des Automaten a_3 auf jeder Komponente seines n-dimensionalen Ausgabevektors nach Formel 4.8 auf den logischen Wert 1 gesetzt und es ergibt sich die Formel

$$g_2(z, x) = \underbrace{1, \ldots, 1}_{n}. \tag{5.4}$$

Es gilt somit für alle Eingaben x, die nicht in einem Zustand z_a oder z_f eingegeben werden oder die zwar in einem Zustand z_a oder z_f eingegeben werden und an dieser Stelle nicht Teil einer zu schützenden Eingabesequenz p sind, dass nur einer der zusätzlichen Automaten pro Komponente des Ausgabevektors dieselbe Ausgabe liefert wie der Automat A_1. Der Ausgabevektor des Automaten A_1 wird weiterhin durch die ursprüngliche Ausgabefunktion $g(z, x) = g_1(z, x)$ bestimmt.
Danach werden die Automaten a_2 und a_3 hinsichtlich ihres Flächenaufwandes optimiert. Wie später in Abschnitt 5.1.3 gezeigt wird, wird im Ergebnis die kombinatorische Logik zur Realisierung der Ausgabefunktionen g_2 und g_3 in den meisten Fällen kleiner sein als die der Ausgabefunktion g_1 des ursprünglichen Automaten A_1. Die Anzahl der Zustandsspeicher wird mit dem Verfahren nicht reduziert. Prinzipiell könnte zwar auch die Anzahl der Zustände verringert werden, da aber die Anzahl der Zustände exponentiell von der Anzahl der Zustandsspeicher in einer solchen Schaltung abhängt, kann durch die Zustandsreduzierung kein wirklicher Effekt erwartet werden. Zur Veranschaulichung dieser Problematik sei erwähnt, dass mit der Halbierung der Anzahl aller Zustände genau ein Zustandsspeicher eingespart werden kann.

5.1.2. Beispiel

In diesem Abschnitt wird das vorgestellte Konzept der spezifischen Fehlertoleranz für sequentielle Schaltungen an einem einfachen Beispielautomaten präsentiert. Der Automat ist in in Abbildung 5.2 dargestellt.
Der ursprüngliche Automat $A = A_1 = (\{0,1\}^1, \{0,1\}^3, \{0,1,2,3,4,5\}, z_0 = 0, f, g)$ hat die Eingabemenge bzw. das Eingabealphabet $X = \{0,1\}^1$, die Ausgabemenge bzw. das Ausgabealphabet $Y = \{0,1\}^3$, sechs Zustände $Z = \{0,1,2,3,4,5\}$ und den Initialzustand $z_0 = 0$. Der ausgezeichnete Zustand, ab dem die zu schützende Eingabesequenz $p = [1,0,0,1,1]$ fehlertolerant verarbeitet werden soll, sei mit $z_a = 1$ bezeichnet.

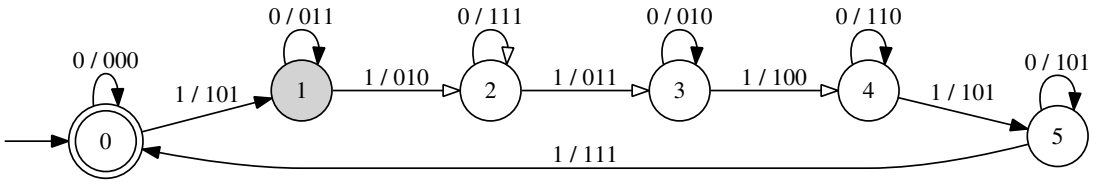

Abbildung 5.2.: Ursprüngliche endliche Zustandsmaschine (Automat A_1)

Wenn die zu schützende Eingabesequenz $[1,0,0,1,1]$ auf den Automaten A_1 im ausgezeichneten Zustand $z_a = 1$ angewendet wird, werden währenddessen auch die Folgezustände $z_f = 2,3,4$ erreicht. Für die Ausgabefunktionen g_2 von a_2 und g_3 von a_3 gilt dann $g_2(1,1) = g_3(1,1) = g_1(1,1) = [010]$, $g_2(2,0) = g_3(2,0) = g_1(2,0) = [111]$, $g_2(2,1) = g_3(2,1) = g_1(2,1) = [011]$, $g_2(3,1) = g_3(3,1) = g_1(3,1) = [100]$. Alle anderen Ausgaben der Funktion $g_2(z,x)$ sind $[000]$ und alle anderen Ausgaben der Funktion $g_3(z,x)$ sind $[111]$.
Dem Automaten A_1 wird somit der Automat a_2 aus Abbildung 5.3

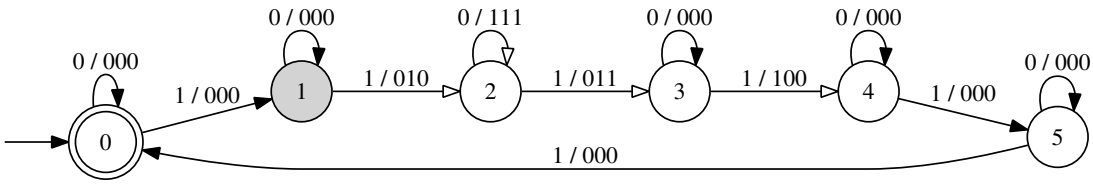

Abbildung 5.3.: Zusätzliche endliche Zustandsmaschine (Automat a_2)

und der Automat a_3 aus Abbildung 5.4 hinzugefügt.
Die Zustandsüberführungsfunktionen aller drei Automaten sind gleich und die Mehrheitsentscheidung über die Ausgaben von A_1, a_2 sowie a_3 wird bitweise durchgeführt.
Anhand der beispielhaft gewählten Eingabesequenz $[1, \mathbf{1,0,0,1,1}, 0, 1, 1]$ wird in Tabelle 5.1 gezeigt, dass die Automaten für die hervorgehobene Teilfolge $p = [1,0,0,1,1]$ fehlertolerant sind sobald sie in den ausgezeichneten Zustand $z_a = 1$ eintreten.

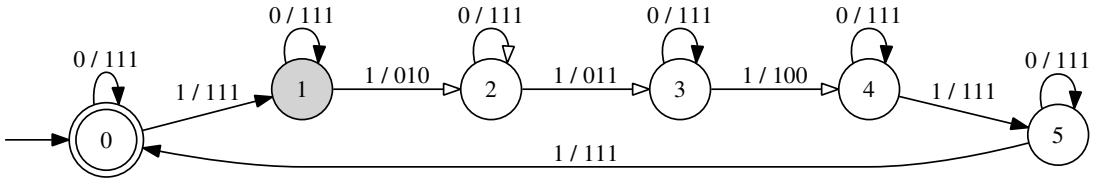

Abbildung 5.4.: Zusätzliche endliche Zustandsmaschine (Automat a_3)

Tabelle 5.1.: Ausgaben der Automaten für die Eingabesequenz $1, \mathbf{1,0,0,1,1}, 0, 1, 1$

Takt	z	x	$f(z,x)$	$g_1(z,x)$	$g_2(z,x)$	$g_3(z,x)$
1	0	1	1	101	000	111
2	1	1	2	010	010	010
3	2	0	2	111	111	111
4	2	0	2	111	111	111
5	2	1	3	011	011	011
6	3	1	4	100	100	100
7	4	0	4	110	000	111
8	4	1	5	101	000	111
9	5	1	0	111	000	111

Nach dem ersten Taktzyklus erreichen die drei Automaten den ausgezeichneten Zustand $z_a = 1$, woraufhin auf sie die zu schützende Eingabesequenz $p = [1, 0, 0, 1, 1]$ angewendet wird. Vom zweiten bis zum sechsten Takt sind die Ausgaben der Automaten A_1, a_2 und a_3 gleich. Das heißt, dass die Zustandsüberführungen $g(1, 1)$, $g(2, 0)$, $g(2, 1)$ und $g(3, 1)$ geschützt sind.
Zu jedem anderen Taktzyklus stimmt die Ausgabe des Automaten A_1 bitweise mit mindestens einer der Ausgaben eines der Automaten a_2 oder a_3 für die gegebene Eingabesequenz komponentenweise überein. Da die Ausgaben bitweise durch einen Voter verarbeitet werden, wird im fehlerfreien Fall die Ausgabe des Automaten A_1 an den Ausgang des Gesamtsystems weitergeleitet.
Zusätzlich zur Eingabesequenz $p = [1, 0, 0, 1, 1]$ werden durch das System auch andere Eingabesequenzen geschützt. Dazu gehören z.B. im Zustand $z_a = 1$ alle Sequenzen, die dem Ausdruck 10^k11 oder 10^k1 zugeordnet werden können, wobei $k \in \mathbb{N}$ ist. Darüber hinaus wird im Zustand 2 z.B. auch jede Teilfolge der Form 0^k1 und 0^k11 geschützt usw.

5.1.3. Experimentelle Ergebnisse

Das kleine Beispiel aus dem vorangegangenen Abschnitt 5.1.2 wurde mit der Hardwarebeschreibungssprache VHDL implementiert und über das Synthesewerkzeug Synopsys mit einer 0.13 μm Standardzellenbibliothek synthetisiert. Um die Signale für einen Reset klar von der Logik zu trennen, die die Eingaben verarbeitet, wurden nur Zustandsregister

mit asynchronem Reset verwendet. Die kombinatorischen Bauteile, die für die Synthese verwendet wurden, sind UND-Gatter mit zwei Eingängen, ODER-Gatter mit zwei Eingängen und Inverter. In den Ergebnissen der Synthese ist die Fläche für den Voter nicht enthalten. Die Ergebnisse sind in Tabelle 5.2 dargestellt.

Tabelle 5.2.: Flächenaufwand des Beispiels in μm^2

Schaltung	gesamt	kombinatorisch	nicht kombinatorisch
A_1	379.2336	290.4768	88.7568
a_2	258.2016	169.4448	88.7568
a_3	298.5456	209.7888	88.7568
gesamt	935.9808	669.7104	266.2704

Beginnend mit dem ausgezeichneten Zustand $z_a = 1$ wurden vier von zwölf möglichen Zustandsübergängen fehlertolerant implementiert. Eine TMR-Implementierung des gesamten Automaten A_1 würde eine Fläche von 1137.7008 μm^2 benötigen. Im Beispiel konnte diese Fläche um 17.73% reduziert werden. Da die Anzahl der Zustandsregister in jedem der Automaten gleich ist, wurde nur die Fläche der kombinatorischen Logik verringert. In diesem Zusammenhang ist die Fläche des kombinatorischen Teils des Beispiels 23.14% kleiner als die Fläche des kombinatorische Teils einer entsprechenden TMR-Implementierung.
Weiter wurde die Methode auf ihre Skalierbarkeit hin untersucht. Beginnend mit dem ausgezeichneten Zustand $z_a = 0$ wurden ein bis zwölf zusammenhängende Zustandsübergänge nach der vorgeschlagenen Methode geschützt. Im ersten Schritt wurde mit dem 0-Übergang $g(0, 0)$ begonnen. Im nächsten Schritt wurde der 1-Übergang $g(0, 1)$ hinzugefügt, dann wieder der 0-Übergang $g(1, 0)$ und so weiter. Das heißt, es wurde immer zwischen den Übergängen der Eingaben 0 und 1 gewechselt und die neu geschützten Zustandsübergänge wurden den bereits vorher geschützten Zustandsübergängen hinzugefügt. Im fünften Schritt hat die Sequenz, die geschützt wird, somit die Form $0^k10^l10^m$ mit $k, l, m \in \mathbb{N}$.
Die Ergebnisse sind in Tabelle 5.3 dargestellt. Die Anzahl der geschützten Übergänge wird als Länge bezeichnet und die benötigte Fläche jedes einzelnen Automaten sowie die des Gesamtsystems ist in μm^2 gegeben. In der Spalte mit der Überschrift "Reduzierung" sind die Flächenreduzierungen im Vergleich zu einem TMR-Ansatz dargestellt, dessen Flächenaufwand durch Verdreifachung der Fläche des Automaten A_1 bestimmt werden kann. Die Spalte mit der Überschrift "gesamt" enthält die Fläche aller kombinatorischen und nicht kombinatorischen Schaltungsteile des Gesamtsystems und die Spalte mit der Überschrift "komb" zeigt die Reduzierung des Flächenaufwandes im Bezug auf den kombinatorischen Teil.
Die Ergebnisse zeigen, dass die vorgeschlagene Methode mit der Anzahl der zu schützenden Zustandsübergänge skaliert. Bis auf die Länge 12 ist das Gesamtsystem immer kleiner als eine TMR-Implementierung. Die Korrelation der vorherigen Ergebnisse aus Tabelle 5.2 mit der Länge 4 aus Tabelle 5.3 ist für die Fläche des Gesamtsystems und

Tabelle 5.3.: Skalierung des Beispiels, beginnend im ausgezeichneten Zustand $z_a = 0$

	Fläche in μm^2				Reduzierung	
Länge	A_1	a_2	a_3	gesamt	gesamt	komb
1	379.2	0.0	242.1	621.3	45.4%	49.1%
2	379.2	225.9	250.1	855.2	24.8%	32.4%
3	379.2	266.3	287.8	933.3	18.0%	23.5%
4	379.2	266.3	303.9	949.4	16.5%	21.6%
5	379.2	295.9	303.9	979.0	13.9%	18.2%
6	379.2	322.8	325.4	1027.4	9.7%	12.7%
7	379.2	309.3	330.8	1019.3	10.4%	13.6%
8	379.2	333.5	355.0	1067.7	6.1%	8.8%
9	379.2	341.6	371.2	1092.0	4.0%	5.3%
10	379.2	347.0	379.2	1105.4	2.8%	3.7%
11	379.2	363.1	379.2	1121.5	1.4%	1.9%
12	379.2	379.2	379.2	1137.6	0%	0%

für den kombinatorischen Teil sehr hoch. Bis auf die Länge 1 konnten die nicht kombinatorischen Teile der zusätzlichen Automaten a_2 und a_3 nicht verkleinert werden. Das liegt daran, dass nur die Ausgabefunktionen der zusätzlichen Automaten verändert wurde und daher die entsprechende Logik, mit der $g_2(z, x)$ und $g_3(z, x)$ realisiert wurden, kleiner ist als die Logik von $g_1(z, x)$.

Die Abbildung 5.5 zeigt eine Gegenüberstellung der Flächenreduzierung des Gesamtsystems und der Reduzierung des kombinatorischen Teils. Von der Länge 1 bis 7 variiert der Unterschied zwischen den beiden Datensätzen von 3% bis 7.6%. Für die Längen 8 bis 12 ist er kleiner als 3%.

Weitere Untersuchungen basieren auf den Benchmark-Schaltungen dk14 und dk15 aus der Benchmark-Suite LGSynth91. Diese endlichen Zustandsmaschinen haben beide einen 3 Bit breiten Eingang und einen 5 Bit breiten Ausgang. Die Schaltung dk14 besitzt 7 Zustände und wird durch 56 Zustandsübergänge beschrieben. Der Automat aus Schaltung dk15 ist wiederum durch 4 Zustände und 32 Zustandsübergänge beschrieben.

Beginnend beim geschützten Zustand $z_a = 1$ wurden zufällig aufeinander folgende Zustandsübergänge ausgewählt, die durch die spezifische Fehlertoleranz geschützt wurden. Die Anzahl der geschützten Zustandsübergänge wurde im Abstand von 10% bezüglich der maximalen Anzahl an Zustandsübergängen gewählt. Sobald die Synthese eines der zusätzlichen Automaten in einer größeren Schaltung als dem ursprünglichen Automaten A_1 resultierte, wurde die größere Schaltung durch eine Kopie von A_1 ersetzt. Die Ergebnisse sind in den Tabellen 5.4 und 5.6 aufgelistet. Zusammen mit den Ergebnissen der Beispielschaltung sind die jeweiligen Flächenreduzierungen der Gesamtsysteme auch zur besseren Übersicht in Abbildung 5.6 dargestellt. Sie enthalten nicht den Mehraufwand des Voters.

Die sich im Vergleich zu einem TMR-Ansatz ergebende Flächenreduzierung liegt nahe

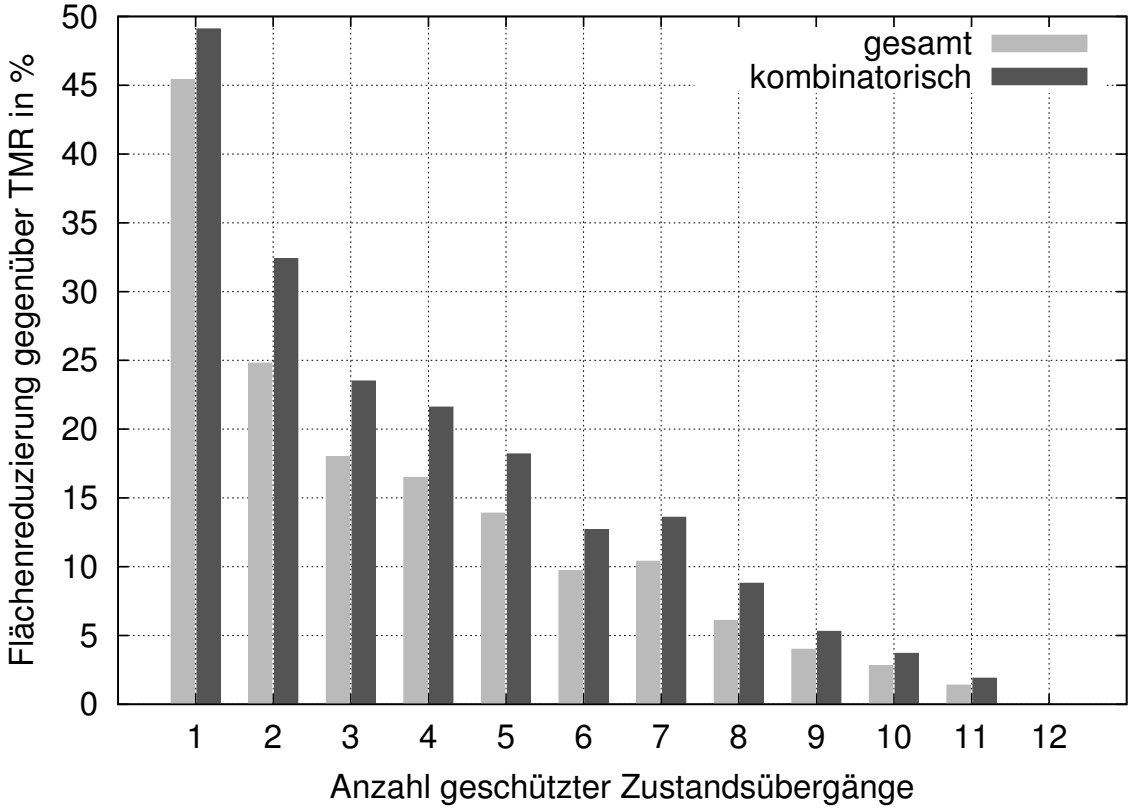

Abbildung 5.5.: Flächenaufwand des sequentiellen Beispiels

Tabelle 5.4.: Messergebnisse der Benchmark-Schaltung dk14

Länge	Fläche in μm^2				Reduzierung	
	A_1	a_2	a_3	gesamt	gesamt	komb
10% (5)	1089.3	761.2	847.2	2697.7	17.4%	16.5%
20% (11)	1089.3	796.1	938.7	2824.1	13.6%	13.9%
30% (16)	1089.3	758.5	895.6	2743.4	16.0%	17.0%
40% (22)	1089.3	763.8	890.3	2743.4	16.0%	17.0%
50% (28)	1089.3	876.8	995.2	2961.3	9.4%	10.8%
60% (33)	1089.3	938.7	1011.3	3039.3	7.0%	8.9%
70% (39)	1089.3	992.5	1089.3	3171.1	3.0%	3.8%
80% (44)	1089.3	1078.5	1089.3	3257.1	0.3%	0.4%
90% (50)	1089.3	1089.3	1089.3	3267.9	0%	0%
100% (56)	1089.3	1089.3	1089.3	3267.9	0%	0%

Tabelle 5.5.: Messergebnisse der Benchmark-Schaltung dk15

Länge	Fläche in μm^2				Reduzierung	
	A_1	a_2	a_3	gesamt	gesamt	komb
10% (3)	693.9	298.5	312.0	1304.4	37.3%	40.8%
20% (6)	693.9	414.2	398.0	1506.1	27.7%	30.2%
30% (9)	693.9	489.5	645.5	1828.9	12.1%	13.3%
40% (12)	693.9	602.5	693.9	1990.3	4.4%	4.8%
50% (16)	693.9	524.5	693.9	1912.3	8.1%	8.9%
60% (19)	693.9	546.0	693.9	1933.8	7.1%	7.8%
70% (22)	693.9	605.2	693.9	1993.0	4.3%	4.7%
80% (25)	693.9	680.5	693.9	2068.3	0.6%	0.7%
90% (28)	693.9	693.9	693.9	2081.7	0%	0%
100% (32)	693.9	693.9	693.9	2081.7	0%	0%

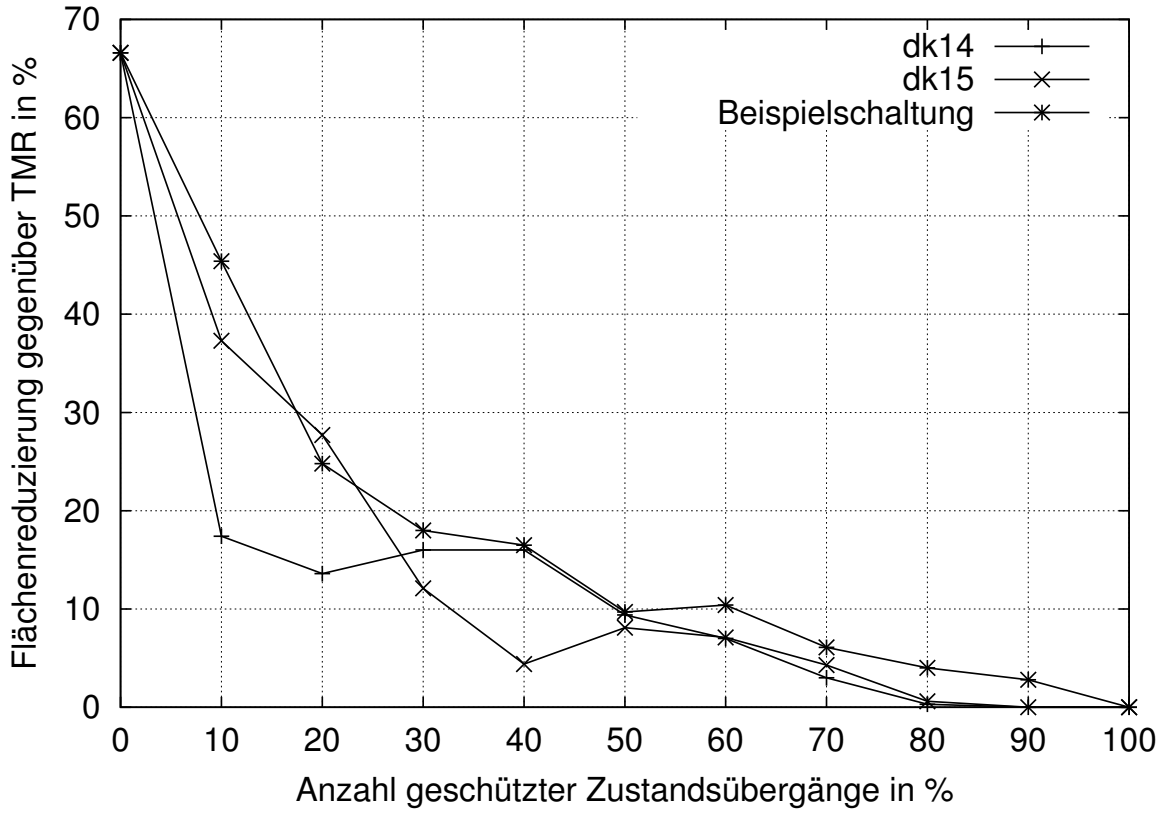

Abbildung 5.6.: Übersicht zur Flächenreduzierung der untersuchten sequentiellen Schaltungen

bei den Ergebnissen des zuvor aufgeführten Beispiels. Für Sequenzen, die bis zu 20% aller Zustandsübergänge überdecken, konnte der Flächenaufwand im Vergleich zu einer TMR-Implementierung um etwa 30% reduziert werden.
Bei weiteren Untersuchungen zeigte sich auch, dass das Verfahren besonders dann zu Einsparungen führt, wenn die Ausgabefunktion des ursprünglichen Automaten aus möglichst vielen verschiedenen Werten aller möglichen Belegungen für den Ausgabevektor y besteht.

5.2. Reduzierung der Zustandsspeicher

In dem vorangegangenen Abschnitt 5.1 wurde gezeigt, wie ein endlicher Automat so entworfen werden kann, dass seine Ausgabefunktion nur im Fall spezieller, vom Entwerfer festgelegter Zustandsübergänge fehlertolerant arbeitet. Der Flächenaufwand der kombinatorischen Logik, die zur Realisierung der teilweise fehlertoleranten Ausgabefunktion nötig ist, wird dadurch in den meisten Fällen reduziert und ist nie größer als bei einem TMR-System. Wie es auch bei TMR-Systemen der Fall ist, besitzt ein fehlertoleranter Automat, der nach dem zuvor beschriebenen Prinzip entworfen wurde, im Vergleich zu seinem nicht fehlertoleranten Ausgangssystem die dreifache Anzahl an Zustandsspeicherelementen.
In diesem Abschnitt wird beschrieben, wie die Anzahl der Zustandsspeicherelemente durch eine Kombination aus der spezifischen Fehlertoleranz für sequentielle Schaltungen und dem Verfahren der Double Modular Redundancy reduziert werden kann. Die Fehlertoleranz der Zustandsüberführungsfunktion f bleibt dabei nach wie vor erhalten und die Ausgabefunktion g arbeitet, wie im bereits vorgestellten Ansatz, genau dann fehlertolerant, wenn ein bestimmter ausgezeichneter Zustand z_a erreicht wurde und die nachfolgenden Eingaben Bestandteil einer zu schützenden Eingabesequenz $p \in P$ sind. Der Automat liefert dann auch wieder für jede Eingabe so lange fehlertolerante Ausgaben, bis die zu schützende Eingabesequenz p verlassen oder ihr Ende erreicht worden ist.
Ein Fehler, der vor dem Erreichen des Zustandes z_a auftritt, wird sich zu diesem Zeitpunkt höchstens auf die Ausgabefunktion auswirken. Da die Zustandsüberführungsfunktion fehlertolerant ausgelegt ist, wird der Zustand z_a und auch alle Folgezustände, die bei der Eingabe einer zu schützenden Eingabesequenz abgelaufen werden, von dem System sicher erreicht.

5.2.1. Schaltungsarchitektur

Damit die Zustandsüberführung eines Automaten $A = A_1$ auch beim Auftreten eines Einzelfehlers korrekt umgesetzt werden kann, wird seine Zustandsüberführungsfunktion f nach dem TMR-Prinzip verdreifacht. Der bei dieser Technik eingesetzte Voter am Ausgang der drei Funktionen $f = f_1 = f_2 = f_3$ bildet aus der Mehrheit seiner Eingaben die aktuelle Zustandskodierung. Diese Kodierung wird üblicherweise in Zustandsspeicherelementen abgelegt, welche in der vorgeschlagenen Schaltungsarchitektur durch fehlertole-

rante Speicherelemente mit rückgekoppeltem Voter entsprechend der Double Modular Redundancy aus Abschnitt 2.3 ersetzt werden.
Auf diese Weise wird im Vergleich zu einem herkömmlichen TMR-Ansatz und auch im Vergleich zu dem zuvor in Abschnitt 5.1 beschriebenen Systemaufbau ein Drittel der Zustandsspeicherelemente eingespart. Gleichzeitig bleibt die Zustandsüberführungsfunktion f aber fehlertolerant gegenüber transienten Fehlern in den Zustandsspeichern. Da die Funktion f bereits fehlertolerant ist, müssen keine zusätzlichen Automaten verwendet werden, um sicherzustellen, dass auch im Fehlerfall der ausgezeichnete Zustand z_a sowie die Folgezustände z_f zum richtigen Zeitpunkt erreicht werden. Die zusätzlichen Automaten a_2 und a_3 in der Systemarchitektur aus Abbildung 5.1 können somit durch rein kombinatorische Schaltungen ersetzt werden. Diese realisieren ausschließlich die Ausgabefunktionen g_2 und g_3 und werden im Folgenden mit s_2 und s_3 bezeichnet.
Da die Ausgabefunktion eines Automaten $g : Z \times X \to Y$ neben der aktuellen Eingabe auch von dem aktuellen Zustand des Systems abhängt, müssen die zusätzlichen kombinatorischen Schaltungen s_2 und s_3 die l-Bits breite Zustandskodierung z aus den Zustandsspeichern bzw. dem nachgeschalteten rückgekoppelten Voter des Automaten A_1 auslesen. Ein entsprechender Systemaufbau ist in Abbildung 5.7 dargestellt.

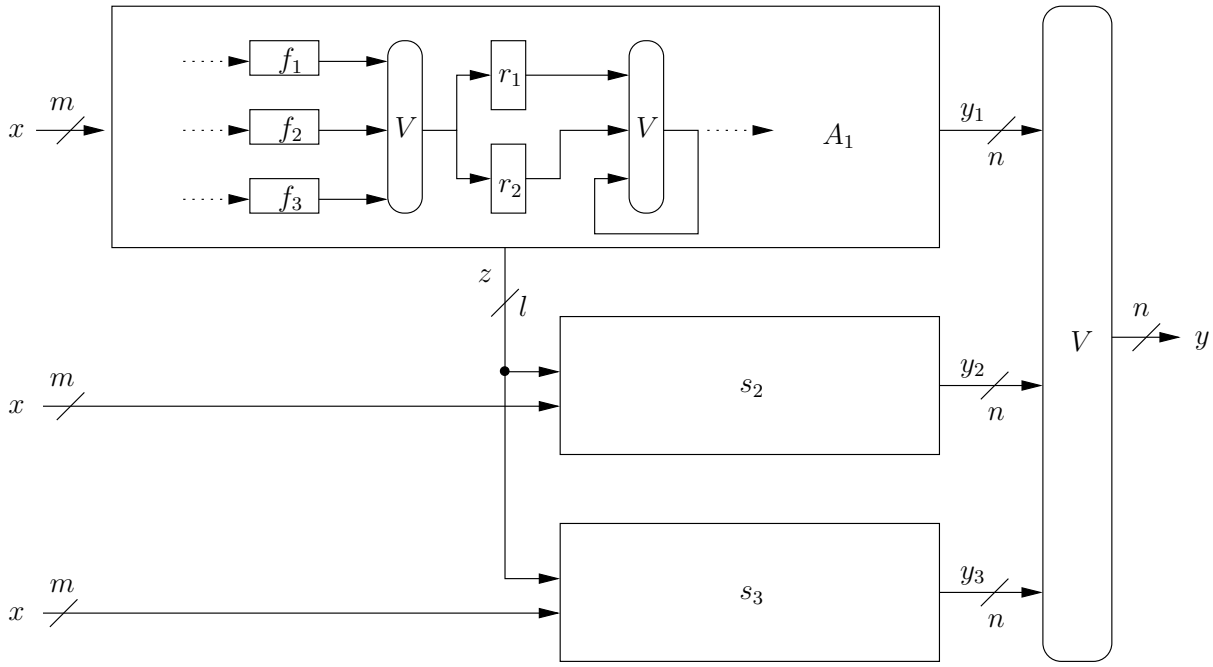

Abbildung 5.7.: Systemaufbau der spezifischen Fehlertoleranz für endliche Automaten mit fehlertoleranten Zustandsspeichern

Der Automat A_1 behält also seine ursprüngliche Funktionalität bei. Es wird lediglich die Zustandsüberführungsfunktion verdreifacht, die Zustandsspeicher durch fehlertolerante Speicherelemente ersetzt und ein zusätzlicher Ausgang eingefügt, an dem die aktuelle Zustandskodierung ausgegeben wird.

5.2.2. Absicherung der Zustandsüberführungsfunktion

Im Vergleich zum bisher existierenden Ansatz der spezifischen Fehlertoleranz für endliche Automaten werden durch den Systemaufbau aus Abbildung 5.7 auch Fehler während eines nicht zu schützenden Teils einer Eingabesequenz vermieden, die dadurch entstehen, dass sich einer der zusätzlichen Automaten in einem falschen Zustand befindet. Das Tolerieren solcher Fehler ist zwar in dem zuvor beschriebenen Ansatz nicht gefordert, wird aber von dem in diesem Abschnitt vorgeschlagenen Systemaufbau mit übernommen. Zur Verdeutlichung dieses Vorteils wird nun ein kurzes Beispiel vorgestellt. Dazu wird zunächst ein Automat mit einer Ausgabewortbreite von drei Bits und eine Eingabesequenz q betrachtet, aus der eine Teilsequenz $p \in P$ nach dem bisherigen Prinzip der spezifischen Fehlertoleranz geschützt werden soll. Es sei angenommen, dass sich die in Tabelle 5.6 dargestellten Ausgaben der Automaten A_1, a_2 und a_3 ergeben, wenn sie sich alle über die gesamte Dauer der Eingabesequenz q in denselben korrekten Zuständen befinden und die Eingabesequenz q im Zustand $z = 1$ eingegeben wird.

Tabelle 5.6.: Ausgaben der Automaten A_1, a_2 und a_3 im jeweils korrekten Zustand

	$z=1$	$z=2$	$z_a=3$	$z_f=4$	$z_f=5$	$z=6$
y_1	101	010	101	111	011	100
y_2	000	000	101	111	011	000
y_3	111	111	101	111	011	111
y	101	010	101	111	011	100

Der kritische Teil $p \in P$ der Eingabesequenz q beginnt im Zustand $z_a = 3$ und endet im Zustand $z_f = 5$. Die Ausgaben $y_1 \dots y_3$ sind entsprechend ihrer Indizes den Automaten $A_1 \dots a_3$ zugeordnet. Die Ausgabe des Voters ist in Zeile y dargestellt.
Es wird nun angenommen, dass sich der Automat a_3 aufgrund eines SEUs in seinem Zustandsregister zu Beginn der Eingabesequenz q nicht im Zustand $z = 1$ befindet. Dann werden von ihm innerhalb der sechs Takte $t = 1, \dots, 6$ beliebige Ausgaben $y_3 = u$ generiert. Wie es in Tabelle 5.7 dargestellt ist, kann dann während eines nicht zu schützenden Teils der Eingabesequenz ein potentieller Fehler φ am Ausgang der Gesamtschaltung entstehen. Ein Beispiel wäre der Wert $y_3 = 101$ im Takt 2, der das fehlerhafte Ergebnis $y = 000$ erzeugen würde.

Tabelle 5.7.: Ausgaben der Automaten, wobei sich a_3 im falschen Zustand befindet

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
y_1	101	010	101	111	011	100
y_2	000	000	101	111	011	000
y_3	u	u	u	u	u	u
y	φ	φ	101	111	011	φ

Eine ähnliche Problematik entsteht auch bei Automaten, die nach dem herkömmlichen TMR-Prinzip entworfen werden. Befindet sich ein Automat in einem falschen Zustand, kann der Automat bis zum nächsten Zurücksetzen seine Fehlertoleranz für nachfolgende Fehler verlieren.
Durch den Systemaufbau aus Abbildung 5.7 werden solche Fehler direkt im Zustandsregister toleriert. Zudem ist die Zustandsüberführungsfunktion dreifach ausgelegt, wodurch auch ein Fehler in der kombinatorischen Logik der Funktion f toleriert wird.

5.2.3. Implementierungsbeispiel

Zur Realisierung des vorgeschlagenen Systemaufbaus muss der ursprünglich gegebene, nicht fehlertolerante Automat A_1 modifiziert werden. Die wesentlichen Änderungen für eine mögliche VHDL-Implementierung werden beispielhaft für eine Schaltung mit 3 Zustandsspeichern in den Abbildungen 5.8 und 5.9 vorgestellt. Die Zustandsüberführungsfunktion f sei in diesem Zusammenhang als separate kombinatorische Schaltungsbeschreibung gegeben.
In Zeile 4 aus Abbildung 5.8 wird zunächst dargestellt, wie die Schaltung um einen zusätzlichen Ausgang `state_out` von der Breite der Anzahl der Zustandsspeicherelemente des ursprünglichen Automaten $A = A_1$ erweitert werden muss. An diesem Ausgang wird die aktuelle Zustandskodierung ausgegeben, welche als Teil der Eingabe für die Schaltungen f, s_2 und s_3 verwendet wird.
Damit in der weiteren VHDL-Beschreibung des Automaten A_1 eine möglichst einfache Mehrheitsentscheidung über die Ausgaben der verdreifachten Zustandsüberführungsfunktionen und die Werte in den Zustandsregistern vorgenommen werden kann, wird die Zustandskodierung explizit angegeben. Ein Beispiel ist in Zeile 10 aus Abbildung 5.8 für den Zustand mit der Kodierung "000" gegeben. Bei komplexeren Automaten können an dieser Stelle auch andere Beschreibungen gewählt werden. Die Mehrheitsentscheidungen in den Zeilen 15-17 und 23-25 aus Abbildung 5.9 müssen dann allerdings noch auf die jeweils verwendeten Datentypen angepasst werden.
Die in den Zeilen 19-25 aus Abbildung 5.8 eingebundene Komponente f realisiert die Zustandsüberführungsfunktion $f : Z \times X \to Z$. Sie erhält entsprechend ihrer Definition an ihrem Eingangsport `in1_f` die aktuelle Eingabe x und an ihrem Eingangsport `s_in_f` den aktuellen Zustand des Automaten A_1. Der Zustand wird von der Schaltung A_1 über den neu eingeführten Ausgang `state_out` bereitgestellt. Die Ausgabe `s_out_f` der Schaltung f trägt die Kodierung des Folgezustandes für den Automaten A_1.
In den Zeilen 4-6 aus Abbildung 5.9 wird beispielhaft eine Instanz der Zustandsüberführungsfunktion f erzeugt. Da diese Funktion verdreifacht wird, ist dieser Anweisungsteil ebenfalls zu verdreifachen. Die Ausgaben der zusätzlichen Instanzen der Zustandsüberführungsfunktion werden dann jeweils den Signalen `next_state_2` und `next_state_3` zugewiesen.
Der Aufbau der Zustandsregister ist exemplarisch in den Zeilen 10-19 aus Abbildung 5.9 gegeben. Essentiell ist dabei die in den Zeilen 15-17 dargestellte Mehrheitsentscheidung über die Ausgaben der drei Instanzen der Zustandsüberführungsfunktion f. Sie bildet zur steigenden Taktflanke den neuen Zustand des Automaten A_1, der durch das

```
1  -- Entity ------------------------------------------
2
3    -- Zustandsausgang
4    state_out : out std_logic_vector(2 downto 0);
5    ...
6
7  -- Deklarationsteil -----------------------------
8
9    -- Zustaende und ihre Kodierung
10   constant s0 : std_logic_vector(2 downto 0) := "000";
11   ...
12
13   -- Zustandssignale
14   signal state, state_1, state_2, next_state_1, next_state_2,
15          next_state_3 : std_logic_vector(2 downto 0) := s0;
16   ...
17
18   -- Funktion f einbinden
19   component f is
20   port(
21     in1_f   : in std_logic_vector(2 downto 0);
22     s_in_f  : in std_logic_vector(2 downto 0);
23     s_out_f : out std_logic_vector(2 downto 0)
24   );
25   end component f;
```

Abbildung 5.8.: Modifikationen im Entity- und Deklarationsteil der VHDL-Beschreibung einer Schaltung A_1

Signal `state_1` repräsentiert wird. Um das DMR-Konzept aus Abbildung 2.3 auf die Zustandsregister anwenden zu können, wird ein weiteres Register nach dem zuvor beschriebenen Prinzip angelegt. Das Signal `state_1` wird dabei durch das Signal `state_2` ersetzt. Der rückgekoppelte Voter wird im Anschluss durch die Zeilen 23-25 aus Abbildung 5.9 erzeugt und die Ausgabe dieses Voters wird dem Automaten A_1 als Kodierung des aktuellen Zustandes durch das Signal `state` bereitgestellt.
Die darauf folgenden Zeilen enthalten die Signalzuweisung für den Ausgangsport des Automaten A_1, der die Kodierung des aktuellen Zustandes des Automaten A_1 anzeigt, und die Beschreibung der Funktion g.
Da die Schaltungen s_2 und s_3 den aktuellen, fehlertolerant erzeugten und gespeicherten Zustand des ursprünglichen Automaten $A = A_1$ direkt aus den Zustandsspeichern des Automaten A_1 auslesen können, realisieren diese Schaltungen ausschließlich die Ausgabefunktion g und können somit als rein kombinatorische Schaltungen implementiert werden. Die Eingabewortbreite der Schaltungen s_2 und s_3 setzt sich dabei jeweils aus der Breite des Zustandsregisters sowie der Eingabewortbreite m des ursprünglichen Automaten $A = A_1$ zusammen.
Die zu schützenden Eingaben $x_p = (z, x) \in p$ werden in beiden Schaltungen s_2 und s_3

```
1  -- Anweisungsteil ------------------------------------
2
3    -- 3 Instanzen der Funktion f erzeugen
4    f_1: f port map ( in1_f   => in1,
5                      s_in_f  => state,
6                      s_out_f => next_state_1 );
7    ...
8
9    -- Zustandsregister fuer state_1 und state_2
10   process (clk, rst)
11   begin
12     if rst = '1' then
13       state_1 <= s0;
14     elsif rising_edge(clk) then
15       state_1 <= ( next_state_1 and next_state_2 ) or
16                  ( next_state_1 and next_state_3 ) or
17                  ( next_state_2 and next_state_3 );
18     end if;
19   end process;
20   ...
21
22   -- rueckgekoppelte Voter
23   state <= ( state_1 and state_2 ) or
24            ( state_1 and state   ) or
25            ( state_2 and state   );
26
27   state_out <= state;
28
29   -- Beschreibung der Funktion g
30   ...
```

Abbildung 5.9.: Modifikationen im Anweisungsteil der VHDL-Beschreibung einer Schaltung A_1

mit denselben Ausgaben festgelegt, wie sie auch vom Automaten A_1 ausgegeben werden. Für alle übrigen Eingaben tragen die zugehörigen Ausgaben der Schaltung s_2 auf jeder Komponente des Ausgabevektors den logischen Wert 0 und in der Schaltung s_3 werden die Komponenten des Ausgabevektors für diese Eingaben entsprechend auf den logischen Wert 1 gesetzt.

5.2.4. Experimentelle Ergebnisse

Das in den vorangegangenen Abschnitten beschriebene Verfahren zur Absicherung der Zustandsüberführungsfunktion f von Automaten, die bezüglich ihrer Ausgabefunktion g nach dem Prinzip der spezifischen Fehlertoleranz für sequentielle Schaltungen entworfen wurden, wurde an den zwei endlichen Zustandsmaschinen dk14 und s1488 aus der Benchmark-Suite LGSynth91 implementiert. Die Schaltung dk14 hat eine Eingabewort-

breite von 3 Bit, eine Ausgabewortbreite von 5 Bit, besitzt 7 Zustände und ist durch 56 Zustandsübergänge vollständig beschrieben. Die Schaltung s1488 hat eine Eingabewortbreite von 8 Bit, eine Ausgabewortbreite von 19 Bit, 48 Zustände und ist durch 251 Zustandsübergänge in Form von Cubes vollständig beschrieben. Die Gesamtzahl der möglichen Zustandsübergänge der Schaltung s1488 beträgt somit 12288.
Bei der Synthese der beiden Schaltungen wurde wieder der Synopsys Design-Compiler und die bereits eingeführte 0.13 μm Standardzellenbibliothek verwendet, aus der ausschließlich UND-Gatter, ODER-Gatter, Inverter, Flip-Flops mit asynchronem Reset und ein Voter-Element genutzt wurden. Die asynchronen Speicherelemente wurden wie zuvor verwendet, um im Hinblick auf die Schaltungsoptimierung die Reset-Signale von der übrigen Logik zu trennen.
Die jeweiligen Teilschaltungen A_1, s_2 und s_3 wurden ohne separate Ausgaberegister implementiert. Ein Ausgaberegister R wurde nur hinter den Voter V aus Abbildung 5.7 geschaltet, der die Ausgabe des Gesamtsystems bestimmt. Als Zustandskodierung wurde in beiden Schaltungen eine binäre Repräsentation der jeweiligen Zustandsnummer gewählt. Die Zustände wurden dabei von 0 beginnend durchnummeriert. Zur Kodierung von 8 Zuständen sind somit 3 Speicherelemente notwendig, zur Kodierung von 16 Zuständen 4 Speicherelemente usw.
Im Bezug auf die Zustandsüberführungsfunktion wurden, wie auch im vorangegangenen Experiment, bei jeder untersuchten Benchmark-Schaltung, ausgehend vom Initialzustand, zufällige, zusammenhängende Zustandsüberführungen geschützt. Die Anzahl der geschützten Zustandsübergänge wurde mit jedem Versuch um 10% erhöht. Die Erhöhung bezieht sich dabei jeweils auf die maximale Anzahl an möglichen Zustandsübergängen pro Schaltung.
Bei den Experimenten wurden außerdem alle Schaltungen, die nach der Synthese mehr Fläche benötigten als die Schaltung, die alle Eingabesequenzen fehlertolerant verarbeitet, durch die Schaltung mit dem vollständigen Schutz für alle Eingabesequenzen ersetzt. Die zu schützenden Zustandsübergänge sind darin ebenfalls enthalten. Dieser Fall kann aufgrund der gewählten Heuristik auftreten, wenn besonders viele Zustandsübergänge geschützt werden sollen und die Ausgabefunktion des Automaten A_1 nur wenige mögliche Belegungen des Ausgabevektors zulässt.
Die experimentellen Ergebnisse sind für die untersuchten Benchmark-Schaltungen in den Tabellen 5.8 und 5.9 aufgelistet. Sie enthalten die Größen der einzelnen Teilsysteme A_1, s_2, s_3, den Aufwand für die Ausgabe-Voter und Ausgaberegister $(V + R)$ sowie die Summe der Flächen aller Teilsysteme.
Neben den Ergebnissen, die durch den vorgeschlagenen Systemaufbau gewonnen wurden, enthalten die Tabellen jeweils eine Zeile mit der Bezeichnung TMR und eine Zeile mit der Bezeichnung TMR-FF. Der Eintrag in der Zeile TMR beschreibt den Flächenaufwand einer herkömmlichen TMR-Realisierung der jeweils untersuchten Benchmark-Schaltung. Die Zeile TMR-FF zeigt wiederum die Fläche des jeweiligen Automaten A_1, die sich ergibt, wenn dieser Automat nach dem Prinzip der spezifischen Fehlertoleranz mit geschützten Zustandsspeichern entworfen wurde. Die DMR-Flip-Flops der Zustandsspeicher wurden dabei durch TMR-Flip-Flops ersetzt und die Voter an den Ausgängen der Logik der Zustandsüberführungsfunktionen wurden weggelassen.

Tabelle 5.8.: Messergebnisse der Benchmark-Schaltung dk14 mit DMR-Zustandsspeicher

geschützte Zustandsübergänge	Fläche in μm^2				
	A_1	s_2	s_3	$V+R$	gesamt
0%	2097.9	0	0	0	2097.9
10%	2097.9	185.6	196.3	209.8	2689.6
20%	2097.9	220.5	282.4	220.6	2821.4
30%	2097.9	150.6	150.6	209.8	2608.9
40%	2097.9	156.0	188.3	209.8	2652.0
50%	2097.9	290.5	371.2	215.1	2974.7
60%	2097.9	320.1	449.2	215.1	3082.3
70%	2097.9	384.6	478.7	209.8	3171.0
80%	2097.9	481.4	481.4	220.6	3281.3
90%	2097.9	481.4	481.4	220.6	3281.3
100%	2097.9	481.4	481.4	220.6	3281.3
TMR	909.1	909.1	909.1	220.6	2947.9
TMR-FF	2127.5				

Tabelle 5.9.: Messergebnisse der Benchmark-Schaltung s1488 mit DMR-Zustandsspeicher

geschützte Zustandsübergänge	Fläche in μm^2				
	A_1	s_2	s_3	$V+R$	gesamt
0%	8539.5	0	0	0	8539.5
10%	8539.5	677.8	796.1	817.3	10830.7
20%	8539.5	984.4	1043.6	817.3	11384.8
30%	8539.5	1202.3	1253.4	817.3	11812.5
40%	8539.5	1237.2	1455.1	822.8	12054.6
50%	8539.5	1487.4	1866.6	828.2	12721.7
60%	8539.5	1920.4	1920.4	828.2	13208.5
70%	8539.5	1920.4	1920.4	828.2	13208.5
80%	8539.5	1920.4	1920.4	828.2	13208.5
90%	8539.5	1920.4	1920.4	828.2	13208.5
100%	8539.5	1920.4	1920.4	828.2	13208.5
TMR	3859.6	3859.6	3859.6	828.2	12407.0
TMR-FF	9176.9				

Da die Ausgaben der verdreifachten Zustandsüberführungsfunktionen direkt in jeweils separate Zustandsregister geschrieben werden, reicht es aus, die Mehrheitsentscheidung über die aktuelle Zustandskodierung erst beim Auslesen der Zustandsspeicher vorzuneh-

men. Die Fehlertoleranz der Zustandsüberführungsfunktion wird dadurch nicht beeinträchtigt.
Die Ergebnisse zeigen, dass die Größen in den Zeilen mit der Bezeichnung $V + R$ leicht mit der Anzahl der geschützten Zustandsübergänge variieren. Der Grund dafür liegt darin, dass ein Voter durch ein einfaches UND-Gatter ersetzt werden kann, wenn eine der Teilschaltungen A_1, s_2 oder s_3 auf der Ausgabekomponente, über die ein Voter entscheiden soll, permanent den Wert 0 erzeugt.
Zur weiteren Auswertung wurden die Messergebnisse einem herkömmlichen TMR-Ansatz der jeweiligen Automaten $A = A_1$ gegenübergestellt. Eine entsprechende Grafik liefert dazu Abbildung 5.10.

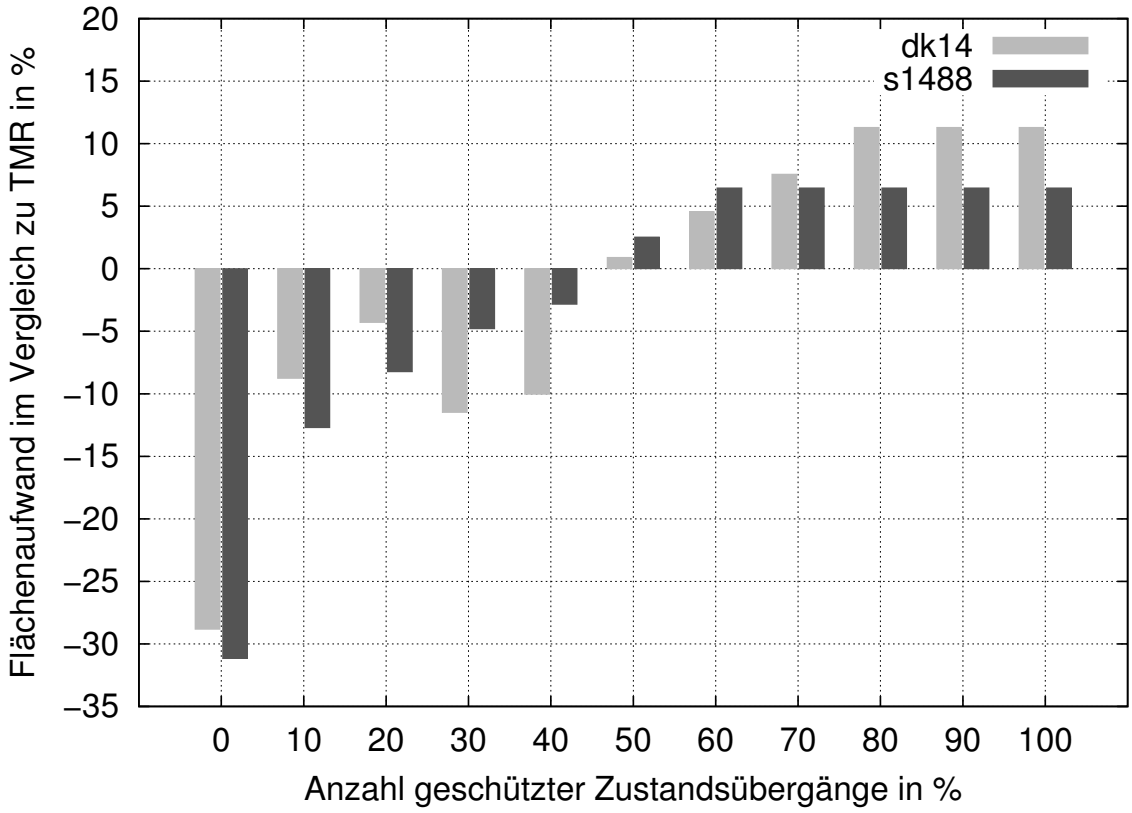

Abbildung 5.10.: Darstellung der Messergebnisse im Vergleich zum TMR-Ansatz

Es zeigt sich im Vergleich mit einer TMR-Realisierung, dass bei den Beispielschaltungen der Schutz von bis zu 40% aller möglichen Zustandsübergänge zu einer Flächenreduzierung von etwa 10% für das Gesamtsystem führen kann. Danach ist ein TMR-Ansatz zwar kleiner, es muss allerdings berücksichtigt werden, dass ein TMR-System im Fall eines Fehlers im Zustandsregister oder in der Zustandsüberführungsfunktion eines der Automaten so lange seine Fehlertoleranz verliert, bis alle drei Automaten wieder synchron arbeiten. Die Synchronisationslogik ist in dem hier vorgeschlagenen Systemaufbau nicht notwendig.

6. Fazit

In der vorliegenden Arbeit wurde ein neues Konzept für den Entwurf fehlertoleranter Digitalschaltungen vorgestellt. Das Verfahren wurde sowohl für kombinatorische als auch für sequentielle Schaltungen ausgearbeitet.
Für kombinatorische Schaltungen stellt die sogenannte spezifische Fehlertoleranz eine neue Technik dar, mit der es erstmals möglich ist, die allgemein sehr hohen Kosten für Maßnahmen zur Realisierung von Fehlertoleranzmechanismen in digitalen Schaltungen gezielt auf eine beliebig wählbare Teilmenge aller möglichen Ein- und Ausgabezuweisungen zu beschränken. Die übrigen Signale der Schaltungen werden nicht fehlertolerant ausgelegt. Der Schutz, der dabei für die gewählte Teilmenge aller Signale bereitgestellt wird, ist ebenso hoch wie der Schutz, der auch durch die dreifach modulare Redundanz gewährleistet wird.
Dieser Ansatz unterscheidet sich grundlegend von den bislang bekannten Verfahren. Erst durch dieses Prinzip besteht die Möglichkeit, Fehlertoleranzmaßnahmen genau auf die real vorhandenen Zuverlässigkeitsanforderungen einer gegebenen Anwendung anpassen zu können. Herkömmliche Verfahren bieten diese Möglichkeit nicht. Sie schützen entweder alle Signale einer Schaltung oder werden lediglich auf Teilbereiche einer Schaltung angewendet.
Wird in einer Schaltung nur ein strukturell beschränkter Schutz implementiert, kann nicht sichergestellt werden, dass die Schaltung für jede Eingabe fehlertolerant arbeitet. Tritt ein Fehler in einem ungeschützten Schaltungsteil auf, kann sich dieser grundsätzlich immer auf die Ausgabe des Gesamtsystems auswirken. Das gilt auch dann, wenn die aktuelle Berechnung kritisch ist. Diese Problematik entsteht bei der spezifischen Fehlertoleranz nicht, da die gewählte Teilmenge an Signalen mit diesem Verfahren durch die gesamte Schaltung hindurch geschützt wird. Entsprechend werden auch alle zugehörigen Funktionen immer fehlertolerant ausgeführt.
Dieses Prinzip ist von Vorteil bei Systemen, die eine Mischung aus kritischen und unkritischen Aufgaben realisieren. Das trifft auf die meisten Anwendungen zu, für die ein gewisses Maß an Zuverlässigkeit gefordert wird. Beispiele finden sich in moderner Fahrzeugelektronik, wo die Zentralverriegelung etwa als unkritischer und eine elektronische Lenkung als kritischer Systembestandteil angesehen werden kann. Werden beide Systeme durch dasselbe Steuergerät kontrolliert, kann mit Hilfe der spezifische Fehlertoleranz garantiert werden, dass die zusätzlich benötigte Hardware zur Absicherung der kritischen Systemaufgaben genau an die Erfordernisse der gegebenen Anwendung angepasst ist. Da für die unkritischen Systemaufgaben keine Fehlertoleranz gefordert werden muss, muss zur Realisierung dieser Aufgaben auch keine zusätzliche Hardware eingesetzt werden.
In verschiedenen Experimenten wurde gezeigt, das dieses Vorgehen beim Entwurf fehlertoleranter Schaltungen erhebliche Kosteneinsparungen mit sich bringt. Im Bezug auf

kombinatorische Schaltungen ergaben die Untersuchungen, dass beim Schutz von 10% bis 50% aller möglichen Ein- und Ausgabezuweisungen durchschnittlich 53% bis 18% des Hardwareaufwands eines TMR-Systems eingespart werden können. Werden ca. 30%-40% aller Ein- und Ausgabezuweisungen fehlertolerant ausgelegt, kann etwa die Fläche eines gesamten Teilsystems einer TMR-Schaltung eingespart werden. Dazu muss jedoch ein sehr rechenintensiver Algorithmus eingesetzt werden, der die zusätzlich bereitgestellten Schaltungen auf Basis von don't-care Werten optimiert.
Da Schaltungsbeschreibungen mit Hilfe von don't-care Werten im industriellen Schaltungsentwurf nicht die Regel sind, es aber ein besonderes Ziel dieser Arbeit war, das vorgeschlagene Verfahren auch für den industriellen Gebrauch einsatzfähig zu machen, wurden verschiedene Heuristiken vorgestellt, mit denen die spezifische Fehlertoleranz auch durch gängige Entwurfswerkzeuge auf beliebig große Schaltungen angewendet werden kann. Zusätzlich wurde ein Framework vorgestellt, mit dem ein Entwerfer eine gegebene Schaltung ohne nennenswerten Aufwand mit der spezifischen Fehlertoleranz ausstatten kann. Die resultierenden Flächeneinsparungen, die sich für die beschriebenen Heuristiken ergaben, liegen lediglich 10% unter den Ergebnissen der rechenintensiven Optimierung von Schaltungsbeschreibungen mit don't-care Werten. D.h. die Flächeneinsparung eines gesamten Teilsystems einer TMR-Schaltung kann mit den Heuristiken beim Schutz von etwa 20% aller möglichen Ein- und Ausgabezuweisungen erreicht werden. Es ist jedoch auch zu berücksichtigen, dass 20% aller möglichen Ein- und Ausgabezuweisungen bei Schaltungen mit großer Eingabewortbreite eine sehr große Anzahl an Vektoren ist, für die die Schaltung fehlertolerant ausgelegt ist.
Im Zusammenhang mit sequentiellen Schaltungen wurde das oben beschriebene Prinzip auf eine beliebig wählbare Teilmenge aller möglichen Eingabesequenzen eines endlichen Automaten abgebildet, die in einem oder mehreren ausgezeichneten Zuständen auf den Automaten angewendet werden. Der Automat liefert für eine solche Eingabesequenz solange fehlertolerante Ausgaben, bis die zu schützende Eingabesequenz beendet ist. Für alle anderen Eingabesequenzen erzeugt der Automat keine fehlertoleranten Ausgaben. Auf diese Weise kann zum Beispiel vermieden werden, dass in sequentiellen Schaltungen auch Ruhezustände mit teuren Fehlertoleranzmechanismen ausgestattet werden.
Zur Realisierung dieses Prinzips wurden in einer ersten Variante drei Automaten mit identischen Zustandsüberführungsfunktionen eingeführt, die bezüglich ihrer Ausgabefunktionen eine der vorgeschlagenen Heuristiken zum gezielten Schutz von Ein- und Ausgabezuweisungen in kombinatorischen Schaltungen implementieren. Die Zustandsüberführungsfunktion des Gesamtsystems ist somit gegen alle Fehler geschützt, die auch durch ein sequentielles TMR-System toleriert werden. Die Ausgabefunktion arbeitet wiederum nur für die gewählten Eingaben fehlertolerant.
Durch experimentelle Ergebnisse wurde gezeigt, dass dieses Prinzip beim Schutz von etwa 20% aller möglichen Zustandsübergänge eine Flächeneinsparung von der Größe einer Teilschaltung eines sequentiellen TMR-Systems mit sich bringt. Ausschlaggebend für die Flächenreduzierung ist einzig und allein die Vereinfachung der Ausgabefunktion der zusätzlichen Automaten. Es zeigte sich auch, dass besonders dann Kosten eingespart werden können, wenn die Ausgabefunktion des ursprünglich gegebenen Automaten möglichst viele verschiedene Belegungen des Ausgabevektors beinhaltet.

In einer zweiten Variante wurde die Fehlertoleranz der Zustandsüberführungsfunktion dadurch erhöht, dass die Zustandsregister des ursprünglichen Automaten verdoppelt und an einen rückgekoppelten Voter angeschlossen wurden. Auf diese Weise werden alle einzelnen transienten Fehler in einem der Speicherelemente toleriert. Die Logik zur Umsetzung der Zustandsüberführungsfunktion wurde weiterhin verdreifacht und auf die Ausgabefunktion wurde wieder das Verfahren der spezifischen Fehlertoleranz angewendet, um die gewählten Eingabesequenzen von den entsprechend spezifizierten Zuständen aus zu schützen.
Für diese Systemarchitektur wurde ein Framework entwickelt, mit dem die spezifische Fehlertoleranz für sequentielle Schaltungen praktisch ohne großen Aufwand mit gängigen Werkzeugen im Schaltungsentwurf umgesetzt werden kann.
Experimentelle Untersuchungen zeigten, dass durch diesen Ansatz bei einem Schutz von 10% bis 40% aller möglichen Zustandsübergänge durchschnittlich etwa 10% der Fläche eines verdreifachten Automaten eingespart werden können. Bei der Berechnung dieses Wertes wurden Maßnahmen zur Synchronisation, die im Fehlerfall bei einem TMR-System notwendig wären, allerdings nicht berücksichtigt. Diese müssten erst noch zusätzlich bei einer TMR-Realisierung implementiert werden, um ein vergleichbares Maß an Fehlertoleranz mit dem in dieser Arbeit vorgeschlagenen Ansatz zu erreichen. Es ist zu erwarten, dass die realen Flächeneinsparungen für die spezifische Fehlertoleranz dann höher ausfallen würden.
Insgesamt stellt die spezifische Fehlertoleranz somit eine effiziente Alternative für den kostengünstigen Entwurf fehlertoleranter Schaltungen dar, da sie im Gegensatz zu partiellen Ansätzen Signale durch die gesamte Schaltung hindurch schützt, mit gängigen Entwurfswerkzeugen sowohl in kombinatorischen als auch in sequentiellen Schaltungen implementiert werden kann und signifikante Einsparungen bezüglich der zusätzlich benötigten Hardware im Vergleich zu TMR-Systemen mit sich bringt.

A. Minimierung kombinatorischer Schaltungen nach Quine - McCluskey

Es wird nachfolgend der Quine-McCluskey Algorithmus nach [McC56] verwendet, um die in Tabelle A.1 dargestellte Funktion zu minimieren. Die Funktion f entspicht der partiell definierten Funktion $s_2(x)$ aus Abschnitt 4.1.1.

Tabelle A.1.: Wertetabelle von f

x_1	x_2	x_3	$f(x)$
0	0	0	0
0	0	1	1
0	1	0	-
0	1	1	1
1	0	0	-
1	0	1	-
1	1	0	-
1	1	1	-

Tabelle A.2.: Wertetabelle von $f_{on/dc}$

x_1	x_2	x_3	$f_{on/dc}(x)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Das Ziel der Minimierung besteht darin, die gegebene Funktion f durch eine möglichst geringe Anzahl an Primimplikanten zu überdecken. Da die Implikanten des Don't-Care-Sets beliebige logische Werte annehmen können, können sie bei der Bestimmung der geeigneten Primimplikanten mit einbezogen werden. Für den Quine-McCluskey Algorithmus wird dazu das Don't-Care Set der Funktion f auf den logischen Wert 1 gesetzt und die folgenden Schritte nacheinander auf die Funktion $f_{on/dc}$ aus Tabelle A.2 angewendet.

1. Schritt: Ermittlung der Fundamentalkonjunktionen

   In diesem Schritt wird zunächst die vollständige disjunktive Normalform der Funktion $f_{on/dc}$ gebildet. Aus dieser werden dann alle Fundamentalkonjunktionen ermittelt, welche anschließend nummeriert und in eine Tabelle eingetragen werden. Die entsprechenden Ergebnisse sind für das Beispiel in dem linken Abschnitt der Tabelle A.3 aufgelistet. Die Fundamentalkonjunktionen sind in Intervallschreibweise dargestellt und aufsteigend in Gruppen mit unterschiedlichen Hamminggewichten eingeteilt worden.

2. Schritt: Verschmelzung von Elementarkonjunktionen

   Die aufgelisteten Intervalle werden nun miteinander verschmolzen. Dabei werden Intervalle benachbarter Gruppen, die sich in maximal einer Belegung voneinander unterscheiden, zu einem neuen Intervall zusammengefasst und in die Tabelle eingetragen. Das Literal mit der abweichenden Belegung wird in dem neuen Intervall auf dont't-care gesetzt und die zugehörigen verschmolzenen Intervalle werden markiert. Dieser Schritt wird so lange wiederholt, bis keine Intervalle mehr miteinander verschmolzen werden können. Für die gegebene Funktion ergibt sich somit der mittlere und rechte Abschnitt aus Tabelle A.3. Die unmarkierten Intervalle sind die Primimplikanten für die Funktion $f_{on/dc}$.

Tabelle A.3.: Bestimmung der Primimplikanten

Fundamentalkonjunktionen			1. Verschmelzung			2. Verschmelzung	
Nr.	Intervall	Marke	Nr.	Intervall	Marke	Nr.	Intervall
1	001	●	1,4	0-1	●	(1,4),(5,7)	--1
2	010	●	1,5	-01	●	(1,5),(4,7)	
3	100	●	2,4	01-	●	(2,4),(6,7)	-1-
4	011	●	2,6	-10	●	(2,6),(4,7)	
5	101	●	3,5	10-	●	(3,5),(6,7)	1--
6	110	●	3,6	1-0	●	(3,6),(5,7)	
7	111	●	4,7	-11	●		
			5,7	1-1	●		
			6,7	11-	●		

3. Schritt: Optimale Überdeckung finden

   Es wird nun bestimmt, wie die Minterme der Funktion f möglichst optimal mit Hilfe der in Schritt 2 gewonnenen Primimplikanten für $f_{on/dc}$ überdeckt werden können. Die Lösung des Überdeckungsproblems ist nicht eindeutig. Sie kann für die gegebene Funktion anhand der Tabelle A.4 nachvollzogen werden. Die erste Spalte enthält die Minterme von f und die Kopfzeile die Primimplikanten für f_{dc}.

Tabelle A.4.: Überdeckungstabelle

	--1	-1-	1--
001	●		
011	●	●	

Es ist zu sehen, dass das Intervall --1 alle Minterme von f überdeckt. Die minimierte Variante der betrachteten Funktion kann somit durch $f(x) = x_3$ beschrieben werden.

B. Übersicht über die verwendeten Benchmark-Schaltungen

B.1. Parameter der kombinatorischen Schaltungen aus der Benchmark-Suite LGSynth91

- Bezeichnung: xor5
  - Eingabewortbreite in Bit: 5
  - Ausgabewortbreite in Bit: 1
  - Anzahl der Vektoren der Schaltungsbeschreibung: 16
- Bezeichnung: Z9sym
  - Eingabewortbreite in Bit: 9
  - Ausgabewortbreite in Bit: 1
  - Anzahl der Vektoren der Schaltungsbeschreibung: 420
- Bezeichnung: t481
  - Eingabewortbreite in Bit: 16
  - Ausgabewortbreite in Bit: 1
  - Anzahl der Vektoren der Schaltungsbeschreibung: 481
- Bezeichnung: parity
  - Eingabewortbreite in Bit: 16
  - Ausgabewortbreite in Bit: 1
  - Anzahl der Vektoren der Schaltungsbeschreibung: 68
- Bezeichnung: max46
  - Eingabewortbreite in Bit: 9
  - Ausgabewortbreite in Bit: 1
  - Anzahl der Vektoren der Schaltungsbeschreibung: 46
- Bezeichnung: o64
  - Eingabewortbreite in Bit: 130
  - Ausgabewortbreite in Bit: 1
  - Anzahl der Vektoren der Schaltungsbeschreibung: 65

B.2. Parameter der sequentiellen Schaltungen aus der Benchmark-Suite LGSynth91

- Bezeichnung: dk14
  - Eingabewortbreite in Bit: 3
  - Ausgabewortbreite in Bit: 5
  - Anzahl der Zustandsübergänge der Schaltungsbeschreibung: 56
  - Anzahl der Zustände: 7
- Bezeichnung: dk15
  - Eingabewortbreite in Bit: 3
  - Ausgabewortbreite in Bit: 5
  - Anzahl der Zustandsübergänge der Schaltungsbeschreibung: 32
  - Anzahl der Zustände: 4
- Bezeichnung: s1488
  - Eingabewortbreite in Bit: 8
  - Ausgabewortbreite in Bit: 19
  - Anzahl der Zustandsübergänge der Schaltungsbeschreibung: 251
  - Anzahl der Zustände: 48

Literaturverzeichnis

[AGK10a] Augustin, Michael ; Gössel, Michael ; Kraemer, Rolf: Eine neue Fehlertoleranzmethode zur Verringerung des Flächenaufwandes von TMR-Systemen. In: *GMM-Fachbericht Band 66: Zuverlässigkeit und Entwurf*. Wildbad Kreuth, Germany, 2010. – ISBN 978–3–8007–3299–9, S. 89–96

[AGK10b] Augustin, Michael ; Gössel, Michael ; Kraemer, Rolf: Reducing the Area Overhead of TMR-Systems by Protecting Specific Signals. In: *Proceedings of the 16th IEEE International On-Line Testing Symposium*. Corfu Island, Greece, 2010. – ISBN 978–1–4244–7724–1, S. 268–273

[AGK10c] Augustin, Michael ; Gössel, Michael ; Kraemer, Rolf: *Elektronische Schaltungsanordnung zum Verarbeiten von binären Eingabewerten*. DE Patent 10 2010 006 383.5 (angemeldet), 2010

[AGK11a] Augustin, Michael ; Gössel, Michael ; Kraemer, Rolf: Effiziente Synthese von Schaltungen mit spezifischer Fehlertoleranz. In: *Tagungsband 23. GI/GMM/ITG-Workshop Testmethoden und Zuverlässigkeit von Schaltungen und Systemen*. Passau, Germany, 2011, S. 93–98

[AGK11b] Augustin, Michael ; Gössel, Michael ; Kraemer, Rolf: Implementation of Selective Fault Tolerance with Conventional Synthesis Tools. In: *Proceedings of the 14th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*. Cottbus, Germany, 2011. – ISBN 978–1–4244–9753–9, S. 213–218

[AGK11c] Augustin, Michael ; Gössel, Michael ; Kraemer, Rolf: Selective Fault Tolerance for Finite State Machines. In: *Proceedings of the 17th IEEE International On-Line Testing Symposium*. Athens, Greece, 2011. – ISBN 978–1–4577–1054–4, S. 49–54

[AGMR02] Aragones, Xavier ; Gonzalez, Jose L. ; Moll, Francesc ; Rubio, Antonio: Noise Generation and Coupling Mechanism in Deep-Submicron ICs. In: *IEEE Design & Test of Computers* 19 (2002), Nr. 5, S. 27–35. – ISSN 0740–7475

[AGSK12] Augustin, Michael ; Gössel, Michael ; Schoof, Gunter ; Kraemer, Rolf: Entwurf fehlertoleranter Zustandsautomaten mit variablem Schutz für spezifische Eingabesequenzen. In: *Tagungsband 24. GI/GMM/ITG-Workshop Testmethoden und Zuverlässigkeit von Schaltungen und Systemen*. Cottbus, Germany, 2012, S. 47–52

[Ait04] Aitken, Rob: New Defect Behavior at 130nm and Beyond. In: *Proceedings of the 9th IEEE European Test Symposium*. Corsica, France, 2004, S. 279–284

[AM03] Almukhaizim, Sobeeh ; Makris, Yiorgos: Fault Tolerant Design of Combinational and Sequential Logic based on a Parity Check Code. In: *Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. Boston, MA, USA, 2003. – ISBN 0–7695–2042–1, S. 563–570

[AVU+08] Abella, Jaume ; Vera, Xavier ; Unsal, Osman S. ; Ergin, Oguz ; Gonzalez, Antonio ; Tschanz, James W.: Refueling: Preventing Wire Degradation due to Electromigration. In: *IEEE Micro* 28 (2008), Nr. 6, S. 37–46. – ISSN 0272–1732

[Bau05] Baumann, Robert C.: Radiation-Induced Soft Errors in Advanced Semiconductor Technologies. In: *IEEE Transactions on Device and Materials Reliability* 5 (2005), Nr. 3, S. 305–316

[BBB+97] Buchner, S. ; Baze, M. ; Brown, D. ; McMorrow, D. ; Melinger, J.: Comparison of Error Rates in Combinational and Sequential Logic. In: *IEEE Transactions on Nuclear Science* 44 (1997), Nr. 6, S. 2209–2216. – ISSN 0018–9499

[Bor99] Borkar, Shekhar: Design Challenges of Technology Scaling. In: *IEEE Micro* 19 (1999), Nr. 4, S. 23–29. – ISSN 0272–1732

[BRC60] Bose, Raj C. ; Ray-Chaudhuri, Dwijendra K.: On A Class of Error Correcting Binary Group Codes. In: *Information and Control* 3 (1960), Nr. 1, S. 68–79

[BSVMH84] Brayton, R. K. ; Sangiovanni-Vincentelli, A. L. ; McMullen, C. T. ; Hachtel, G. D.: *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA, USA : Kluwer Academic Publishers, 1984

[CA08] Chandra, Vikas ; Aitken, Robert: Impact of Technology and Voltage Scaling on the Soft Error Susceptibility in Nanoscale CMOS. In: *Proceedings of the 23rd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. Cambridge, MA, USA, 2008. – ISSN 1550–5774, S. 114–122

[CH84] Chen, C. L. ; Hsiao, M. Y.: Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review. In: *IBM Journal of Research and Development* 28 (1984), Nr. 2, S. 124–134

[Con02] Constantinescu, Cristian: Impact of Deep Submicron Technology on Dependability of VLSI Circuits. In: *Proceedings of the 2002 International*

Conference on Dependable Systems and Networks. Washington, DC, USA : IEEE Computer Society, 2002 (DSN'02). – ISBN 0–7695–1597–5, S. 205–209

[CRM05] Cazeaux, José M. ; Rossi, Daniele ; Metra, Cecilia: Self-Checking Voter for High Speed TMR Systems. In: *Journal of Electronic Testing* 21 (2005), Nr. 4, S. 377–389. – ISSN 0923–8174

[Dan94] Dankmeier, Wilfried: *Codierung: Fehlerbeseitigung und Verschlüsselung*. Braunschweig/Wiesbaden, Germany : Vieweg, 1994 (DUD Fachbeiträge). – ISBN 3–528–05399–2

[DNR02] Dupont, Eric ; Nicolaidis, Michael ; Rohr, Peter: Embedded Robustness IPs for Transient-Error-Free ICs. In: *IEEE Design & Test of Computers* 19 (2002), Nr. 3, S. 56–70. – ISSN 0740–7475

[GKV10] Gleichner, Christian ; Koal, Tobias ; Vierhaus, Heinrich T.: Effiziente Verfahren der Selbstreparatur von Logik. In: *Tagungsband 22. GI/GMM/ITG-Workshop Testmethoden und Zuverlässigkeit von Schaltungen und Systemen*. Paderborn, Germany, 2010, S. 79–84

[GOSM08] Gössel, Michael ; Ocheretny, Vitaly ; Sogomonyan, Egor ; Marienfeld, Daniel: *New Methods of Concurrent Checking*. Cambridge, UK : Springer, 2008 (Frontiers in Electronic Testing). – ISBN 978–1–4020–8419–5

[Ham50] Hamming, Richard W.: Error Detecting and Error Correcting Codes. In: *The Bell System Technical Journal* 26 (1950), Nr. 2, S. 147–160

[HP03] Huffman, W. C. ; Pless, Vera: *Fundamentals of Error Correcting Codes*. Cambridge, UK : Cambridge University Press, 2003. – ISBN 0–521–78280–5

[Hsi70] Hsiao, M. Y.: A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes. In: *IBM Journal of Research and Development* 14 (1970), Nr. 4, S. 395–401

[Joh88] Johnson, Barry W.: *Design & Analysis of Fault Tolerant Digital Systems*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1988. – ISBN 0–201–07570–9

[KJ10] Kohavi, Zvi ; Jha, Niraj K.: *Switching and Finite Automata Theory*. Third. Cambridge University Press, 2010. – ISBN 978–0–521–85748–2

[KOWS06] KleinOsowski, Aj ; Oldiges, Phil ; Williams, Richard Q. ; Solomon, Paul M.: Modeling Single-Event Upsets in 65-nm Silicon-on-Insulator Semiconductor Devices. In: *IEEE Transactions on Nuclear Science* 53 (2006), Nr. 6, S. 3321–3328. – ISSN 0018–9499

[KZK+98] Kim, Ilyoung ; Zorian, Yervant ; Komoriya, Goh ; Pham, Hai ; Higgins, Frank P. ; Lewandowski, Jim L.: Built in self repair for embedded high density SRAM. In: *Proceedings of the International Test Conference 1998*. Washington (DC), USA, 1998. – ISBN 0–7803–5093–6, S. 1112–1119

[Lal01] Lala, Parag K.: *Self-Checking and Fault-Tolerant Digital Design*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2001. – ISBN 0–12–434370–8

[Mas96] Massengill, Lloyd W.: Cosmic and Terrestrial Single-Event Radiation Effects in Dynamic Random Access Memories. In: *IEEE Transactions on Nuclear Science* 43 (1996), Nr. 2, S. 567–593. – ISSN 0018–9499

[MB59] Muller, D. E. ; Bartky, W. S.: A Theory of Asynchronous Circuits. In: *Proceedings of the International Symposium on the Theory of Switching*, Harward University Press, 1959, S. 204–243

[McC56] McCluskey, Edward J.: Minimization of Boolean Functions. In: *The Bell System Technical Journal* 35 (1956), S. 1417–1444

[MM05] Mitra, Subhasish ; McCluskey, Edward J.: *Word Voter for Redundant Systems*. US Patent 6,910,173, 2005

[MT03] Mohanram, Kartik ; Touba, Nur A.: Partial Error Masking to Reduce Soft Error Failure Rate in Logic Circuits. In: *Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. Boston, MA, USA, 2003. – ISBN 0–7695–2042–1, S. 433–440

[MZK09] Mitra, Subhasish ; Zhang, Ming ; Kim, Kee S.: *System and Shadow Bitstable Circuits Coupled to Output Joining Circuit*. US Patent 7,523,371, 2009

[Neu56] Neumann, John von: Probabilistic logics and synthesis of reliable organisms from unreliable components. In: Shannon, C. (Hrsg.): *Automata Studies*, Princeton University Press, 1956, S. 43–98

[Nic10] *Kapitel* 2. In: Nicolaidis, Michael: *Soft Errors in Modern Electronic Systems*. New York, NY, USA : Springer, 2010 (Frontiers in Electronic Testing). – ISBN 978–1–4419–6992–7, S. 38–39

[PH11] Polian, Ilia ; Hayes, John P.: Selective Hardening: Toward Cost-Effective Error Tolerance. In: *IEEE Design & Test of Computers* 28 (2011), Nr. 3, S. 54–63. – ISSN 0740–7475

[Pra96] Pradhan, Dhiraj K.: *Fault-Tolerant Computer System Design*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1996. – ISBN 0–13–057887–8

[RS09] Reichardt, Jürgen ; Schwarz, Bernd: *VHDL-Synthese: Entwurf digitaler Schaltungen und Systeme*. Oldenbourg Wissenschaftsverlag GmbH, 2009. – ISBN 978–3–486–58987–0

[Sch10] Schlünder, Christian: Design for Reliability (DfR) - A key requirement for modern product design. In: *GMM-Fachbericht Band 66: Zuverlässigkeit und Entwurf*. Wildbad Kreuth, Germany, 2010. – ISBN 978–3–8007–3299–9, S. 89–96

[SKJW07] Schoof, Gunter ; Kraemer, Rolf ; Jagdhold, Ulrich ; Wolf, Christoph: Fault-Tolerant Design for Applications Exposed to Radiation. In: *Proceedings of the DASIA 2007 – Data Systems in Aerospace – Conference*. Naples, Italy, 2007. – ISBN 92–9092–202–8

[SKK+02] Shivakumar, Premkishore ; Kistler, Michael ; Keckler, Stephen W. ; Burger, Doug ; Alvisi, Lorenzo: Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. In: *Proceedings of the 2002 International Conference on Dependable Systems and Networks*. Washington, DC, USA : IEEE Computer Society, 2002 (DSN'02). – ISBN 0–7695–1101–5, S. 389–398

[Sog76] Sogomonyan, Egor S.: Design of Self-Repair Systems by Using Self-Checking Circuits. In: *Proceedings of the 6th International Symposium on Fault-Tolerant Computing*. Pittsburgh, PA, USA, 1976, S. 39–44

[SS09] She, Xiaoxuan ; Samudrala, P.K.: Selective Triple Modular Redundancy for Single Event Upset (SEU) Mitigation. In: *Proceedings of the 2009 NASA/ESA Conference on Adaptive Hardware and Systems*. San Francisco, CA, USA, 2009. – ISBN 978–0–7695–3714–6, S. 344–350